WebSphere® Application Server – Express, Version 6

Administering applications and their environment

Note

Before using this information, be sure to read the general information under “Notices” on page 2073.

Compilation date: January 20, 2005

Contents

How to send your comments . . . ix

Chapter 1. Overview and new features for administering applications and their environments . . 1
Overview of administering applications and their environments . . . 2
Getting started with WebSphere Application Server . . . 2
Introduction: System administration . . . 3
  Introduction: Administrative console . . . 4
  Introduction: Administrative scripting (wsadmin) . . . 4
  Introduction: Administrative commands . . . 5
  Introduction: Administrative programs. . . . 5
  Introduction: Administrative configuration data . . . 5
Introduction: Servers . . . 5
  Introduction: Application servers . . . 6
  Introduction: Web servers . . . 7
Introduction: Environment . . . 7

Chapter 2. How do I administer applications and their environments? . . . 9

Chapter 3. Setting up the application serving environment . . . 15
Planning the installation (diagrams) . . . 15
  Planning to install WebSphere Application Server - Express . . . 17
  Planning to install Web server plug-ins . . . 22
  Planning to install WebSphere Application Server Clients . . . 25
  Planning to create application server environments . . . 26
  Queuing network. . . . 27
Configuring the product after installation . . . 32
  firststeps command . . . 33
  Using the Profile creation wizard . . . 36
  wasprofile command . . . 43
  Using the installation verification test . . . 53
Configuring ports . . . 55
  Port number settings in WebSphere Application Server versions . . . 56
Communicating with Web servers . . . 57
  Web server plug-in properties settings . . . 59
  Web server plug-in custom properties . . . 66
  Web server plug-in configuration service properties settings . . . 67
  Application Server property settings for a Web server plug-in . . . 67
  Web server plug-in configuration properties . . . 69
  Web server plug-in connections . . . 71
  Web server plug-in remote user information processing . . . 72
  Web server plug-ins . . . 72
  Checking your IBM HTTP Server version . . . 73
  Web server tuning parameters. . . . 73
  Gskit install images files . . . 77
  Plug-ins: Resources for learning . . . 77
  Web server plug-in tuning tips . . . 77
  Tuning Web servers . . . 79
Setting up the administrative architecture . . . 79
  Administration service settings. . . . 79
  Administrative agents: Resources for learning . . . 86
Configuring the environment . . . 86
  Virtual hosts . . . 87
  Configuring virtual hosts . . . 89

Variables 93
Configuring WebSphere variables 94
Shared library files 97
Managing shared libraries 97
Environment: Resources for learning 104
Working with server configuration files 104
Configuration documents 104
Configuration document descriptions 106
Object names 108
Configuration repositories 108
Handling temporary configuration files resulting from session timeout 108
Changing the location of temporary configuration files 109
Changing the location of backed-up configuration files 109
Changing the location of temporary workspace files 110
Backing up and restoring administrative configurations 110
Transformation of configuration files 110
Server configuration files: Resources for learning 110
Administering application servers 111
Application servers 111
Creating application servers 112
Managing application servers. 113
Creating generic servers 124
Configuring transport chains 126
Custom services 142
Developing custom services 142
Process definition 145
Defining application server processes 145
Java virtual machines (JVMs) 155
Using the JVM 155
Preparing to host applications 162
Java memory tuning tips 163
Tuning application servers 167
Web services client to Web container optimized communication 168
Application servers: Resources for learning 169

Chapter 4. Using the administrative clients 171
Using the administrative console 171
Starting and stopping the administrative console 171
Setting the session timeout for the administrative console 174
Administrative console areas 175
Specifying console preferences 180
Accessing help and product information from the administrative console 184
Administrative console: Resources for learning 184
Using scripting (wsadmin) 185
Getting started with scripting 186
Deploying applications using scripting 254
Managing deployed applications using scripting 256
Configuring servers with scripting 283
Configuring connections to Webservers with scripting. 300
Managing servers with scripting. 301
Configuring security with scripting 310
Configuring data access with scripting 312
Configuring messaging with scripting 328
Configuring mail, URLs, and resource environment entries with scripting. 341
Troubleshooting with scripting 354
Scripting reference material 358

Using Ant to automate tasks . . . 626
ws_ant command . . . 627
Ant tasks for deployment and server operation . . . 627
Ant tasks for building application code . . . 627
Using administrative programs (JMX) . . . 627
Creating a custom Java administrative client program using WebSphere Application Server administrative Java APIs . . . 628
Extending the WebSphere Application Server administrative system with custom MBeans . . . 633
Developing administrative programs for multiple Java 2 Platform, Enterprise Edition application servers . . . 638
Deploying and managing a custom Java administrative client program with multiple Java 2 Platform, Enterprise Edition application servers . . . 641
Migrating Java Management Extensions V1.0 to Java Management Extensions V1.2 . . . 642
Java Management Extensions interoperability . . . 642
Managing applications through programming . . . 643
Using command line tools . . . 668
Example: Security and the command line tools . . . 669
startServer command . . . 669
stopServer command . . . 670
startManager command . . . 671
stopManager command . . . 673
startNode command . . . 674
stopNode command . . . 675
addNode command . . . 677
serverStatus command . . . 681
removeNode command . . . 682
cleanupNode command . . . 683
syncNode command . . . 684
backupConfig command . . . 685
restoreConfig command . . . 686
EARExpander command . . . 687
GenPluginCfg command . . . 688

Chapter 5. Starting and stopping quick reference . . . 691

Chapter 6. Class loading . . . 693
Class loaders . . . 693
Configuring class loaders of a server . . . 697
Class loader collection . . . 698
Class loader ID . . . 698
Class loader mode . . . 698
Class loader settings . . . 699
Configuring application class loaders . . . 699
Configuring Web module class loaders . . . 700
Configuring class preloading . . . 701
Class loading: Resources for learning . . . 702

Chapter 7. Deploying and administering applications . . . 705
Enterprise (J2EE) applications . . . 705
System applications . . . 705
Installing application files . . . 706
Installable module versions . . . 706
Ways to install applications or modules . . . 707
Installing application files with the console . . . 709
Example: Installing an EAR file using the default bindings . . . 719
Installing J2EE modules with JSR-88 . . . 720

Customizing modules using DConfigBeans 721
Enterprise application collection 722
Name 723
Status 723
Enterprise application settings 723
Configuring an application 727
Application bindings 728
Mapping modules to servers 732
Starting and stopping applications 733
Disabling automatic starting of applications 734
Target mapping collection 734
Exporting applications 736
Exporting DDL files 736
Updating applications 736
Ways to update application files. 739
Preparing for application update settings 740
Hot deployment and dynamic reloading 744
Uninstalling applications 752
Removing a file. 752
Deploying and administering applications: Resources for learning 753

Chapter 8. Developing WebSphere applications. 755
Web applications 755
Task overview: Developing and deploying Web applications 755
Task overview: Managing HTTP sessions 792
Modifying the default Web container configuration 803
Configuring session management by level 808
Configuring session tracking 809
Configuring session tracking for Wireless Application Protocol (WAP) devices 813
Configuring for database session persistence. 814
EJB applications 817
Task overview: Using enterprise beans in applications 817
Using access intent policies 822
Managing EJB containers 831
Deploying EJB modules 837
Client applications. 839
Using application clients 839
Running application clients 852
Installing Application Client for WebSphere Application Server 869
Deploying J2EE application clients on workstation platforms 874
Web services 952
Implementing Web services applications 952
Deploying Web services 964
Configuring Web service client bindings 966
Configuring the scope of a Web service port 969
Web Services Invocation Framework (WSIF): Enabling Web services 971
Getting started with UDDI Registry. 975
Planning to use Web services 977
Setting up and deploying a new UDDI Registry 981
Publishing WSDL files 992
Configuring endpoint URL information for JMS bindings 995
Configuring Web services applications with the wsadmin tool 997
WSIF system management and administration 997
Using the UDDI Registry 1004
Removing and reinstalling UDDI Registry 1108
Configuring the UDDI Registry Application 1110

Managing the UDDI Registry . . . 1114
Data access resources . . . 1126
Task overview: Accessing data from applications . . . 1126
Deploying data access applications . . . 1156
Messaging resources . . . 1242
Learning about messaging with WebSphere Application Server . . . 1242
Installing and configuring a JMS provider . . . 1252
Using asynchronous messaging . . . 1253
Maintaining Version 5 default messaging resources . . . 1253
Using JMS resources of WebSphere MQ . . . 1286
Using JMS resources of a generic provider . . . 1353
Administering support for message-driven beans . . . 1361
Mail, URLs, and other J2EE resources. . . . 1378
Using mail . . . 1378
Using URL resources within an application . . . 1382
Resource environment entries . . . 1385
Configuring mail providers and sessions . . . 1389
Security . . . 1393
Securing applications and their environments . . . 1393
Planning to secure your environment . . . 1394
Implementing security considerations at installation time . . . 1404
Integrating IBM WebSphere Application Server security with existing security systems . . . 1409
Administering security . . . 1417
Deploying secured applications . . . 1767
Naming and directory . . . 1779
Using naming . . . 1779
Configuring name servers . . . 1789
Configuring and viewing name space bindings . . . 1790
Developing applications that use JNDI . . . 1794
Developing applications that use CosNaming (CORBA Naming interface) . . . 1809
Object Request Broker . . . 1813
Managing Object Request Brokers . . . 1813
Transactions . . . 1831
Configuring transaction properties for an application server . . . 1831
Using the transaction service . . . 1840
Managing active and prepared transactions . . . 1850
Managing transaction logging for optimum server availability. . . . 1851
Interoperating transactionally between application servers. . . . 1854
Using one-phase and two-phase commit resources in the same transaction . . . 1854
WebSphere programming extensions . . . 1858
ActivitySessions . . . 1858
Application profiling . . . 1872
Asynchronous beans . . . 1884
Dynamic cache . . . 1901
Dynamic query . . . 1949
Internationalization . . . 1978
Object pools . . . 1988
Scheduler . . . 1994
Startup beans . . . 2019
Work area . . . 2020

Chapter 9. Troubleshooting deployment . . . 2047
Errors or problems deploying, installing, or promoting applications . . . 2047
Troubleshooting testing and first time run problems . . . 2051
Errors starting an application . . . 2052
The application does not start or starts with errors . . . 2056

A web resource does not display 2057
Cannot uninstall an application or remove a node or application server 2059

Chapter 10. Troubleshooting administration 2061
Administration and administrative console troubleshooting tips 2061
Installation completes but the administrative console does not start 2063
Errors connecting to the administrative console from a browser 2064
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. 2064
A user on Mozilla browser Version 1.4 selects a check box on a collection table, presses Enter, and receives an error. 2064
A user on Mozilla browser Version 1.4 enters an invalid ID or password, presses Enter, and receives an error message 2065
Web server plug-in troubleshooting tips 2065
The server process does not start or starts with errors 2067
The stopServer.sh hangs and creates a Java core dump (Red Hat Linux) 2068
Problems starting or using the wsadmin command 2068
Problems using tracing, logging or other troubleshooting features 2071

Notices 2073

Trademarks and service marks 2075

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

This topic summarizes the contents and organization of the administration documentation, including links to conceptual overviews and descriptions of new features.

- Overview of administering applications and their environments
- What is new for administrators

Sections in the administration documentation:

Chapter 3, “Setting up the application serving environment,” on page 15
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″ in the information center.

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.

Chapter 4, “Using the administrative clients,” on page 171
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 5, “Starting and stopping quick reference,” on page 691
This section summarizes what can be started and stopped, including applications and the application servers on which these applications are deployed.

Chapter 6, “Class loading,” on page 693
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 7, “Deploying and administering applications,” on page 705
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 9, “Troubleshooting deployment,” on page 2047
This section describes how to identify and handle a variety of problems encountered during development, assembly, and deployment activities.

Chapter 10, “Troubleshooting administration,” on page 2061
This section describes how to identify and handle a variety of problems encountered during administrative activities.

Overview of administering applications and their environments

This topic provides links to conceptual overviews of administering your applications and application serving environment.

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 Education on Demand

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
  - Custom Java administrative client (JMX)
- Topologies and logical administrative domains
  - Resource scoping
- Applications and application resources
  - Application management overview
  - JDBC
  - Installing and uninstalling applications
  - Managed application resources - Enhanced EAR files
  - Fine grained application updates
- Servers
  - Server templates
  - Custom services
  - Manage Web server nodes
- Configuration management
  - Configuration repository
  - Configuration archives
  - File synchronization

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.

IBM WebSphere Application Server products provide a next-generation application server on an industry-standard foundation. Each product addresses a distinct set of scenarios and needs. WebSphere Application Server, Version 6 product offerings are described in ″Packaging″ in the information center.

Planning

See ″Task overview: Installing″ in the information center for a description of typical scenarios for each WebSphere Application Server product.

Installing

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

Configuring

See ″Using the Profile creation wizard″ in the information center for a description of installing other stand-alone Application Servers on your machine.

Migrating

See ″Migration and coexistence overview″ and ″Migrating and coexisting″ in the information center 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 a way to sample WebSphere Application Server functionality by quickly deploying Web components, such as servlets and JSP files. The method is not recommended as an official development method. See ″Fast paths for WebSphere Application Server″ in the information center to get started.

Introduction: System administration

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.
- **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 4.
- **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 5.

- **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 5.
- **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 5.

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.

See the *Using the administrative clients* PDF for information on how you begin using the console. See also the **Reference > Administrator > Settings** section of the Information Center navigation. It lists the settings or properties you can configure.

Introduction: Administrative scripting (wsadmin)

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

The following list highlights the topics and tasks available with scripting. See the *Administering applications and their environment* PDF for more information on how to perform these tasks.

- Getting started with scripting Provides an introduction to WebSphere Application Server scripting and information about using the wsadmin tool. Topics include information about the scripting languages and the scripting objects, and instructions for starting the wsadmin tool.
- Deploying applications Provides instructions for deploying and uninstalling applications. For example, stand-alone Java archive files and Web archive files, the administrative console, remote enterprise archive (EAR) files, file transfer applications, and so on.
- Managing deployed applications Includes tasks that you perform after the application is deployed. For example, starting and stopping applications, checking status, modifying listener address ports, querying application state, configuring a shared library, and so on.
- Configuring servers Provides instructions for configuring servers, such as creating a server, modifying and restarting the server, configuring the Java virtual machine, disabling a component, disabling a service, and so on.

- Configuring connections to Web servers Includes topics such as regenerating the plug-in, creating new virtual host templates, modifying virtual hosts, and so on.
- Managing servers Includes tasks that you use to manage servers. For example, stopping nodes, starting and stopping servers, querying a server state, starting a listener port, and so on.
- Configuring security Includes security tasks, for example, enabling and disabling 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, reference tables, URL providers, URLs, and so on.
- Dynamic caching Includes caching topics, for example, creating, viewing and modifying a cache instance.
- Troubleshooting Provides information about how to troubleshoot using scripting. For example, tracing, thread dumps, profiles, and so on.
- Obtaining product information Includes tasks such as querying the product identification.
- Scripting reference material Includes all of the reference material related to scripting. Topics include the syntax for the wsadmin tool and for the administrative command framework, explanations and examples for all of the scripting object commands, the scripting properties, and so on.

Introduction: Administrative commands

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

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

Introduction: Administrative configuration data

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

Introduction: Servers

Application servers

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

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

Introduction: Application servers

Overview

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

To begin creating and managing an application server, see “Administering application servers” on page 111.

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” on page 126 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” on page 142 for more information.
- Define command-line information that passes to a server when it starts or initializes. See the *Using the administrative clients* PDF for more information.
- “Tuning application servers” on page 167
- Enhance the performance of the application server JVM. See “Using the JVM” on page 155 for more information.
- Use an Object Request Broker (ORB) for RMI/IIOP communication. See the *Developing and deploying applications* PDF for more information.

Asynchronous messaging

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

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

Generic Servers

In distributed platforms, the Generic Servers feature allows you create a generic server as an application server instance within the WebSphere Application Server administration, and associate it with a non-WebSphere server or process. The generic server can be associated with any server or process necessary to support the application server environment, including:

- A Java server
- A C or C++ server or process
- A CORBA server
- A Remote Method Invocation (RMI) server

After you define a generic server, you can use the Application Server administrative console to start, stop, and monitor the associated non-WebSphere server or process when stopping or starting the applications that rely on them.

For more information, refer to “Creating generic servers” on page 124.

Introduction: Web servers

In the WebSphere Application Server product, an application server works with a Web server to handle requests for dynamic content, such as servlets, from Web applications. Go to http://www.ibm.com/software/webservers/appserv/doc/latest/prereq.html for the most current information about supported Web servers.

The application server and Web server communicate using “Web server plug-ins” on page 72. “Communicating with Web servers” on page 57 describes how to set up your Web server and Web server plug-in environment and how to create a Web server definition. The Web server definition associates a Web server with an application server. After you create a Web server definition, you can use the administrative console to perform the following functions for that Web server:

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

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

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

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

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

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

Refer to your Web server documentation for information on how to administer that Web server. For tips on tuning your Web server plug-in, see “Web server plug-in tuning tips” on page 77.

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.

Web servers

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

For more information, refer to “Introduction: Web servers.”

Variables

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

For more information, refer to “Configuring WebSphere variables” on page 94.

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

Create WebSphere profiles

Profiles are the files that define a stand-alone Application Server node, a managed node, or a deployment manager node. A profile also includes all of the files that the node can change.

Documentation	Show me:	Tell me
	- Standalone node	

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	Tell me:
	• Repository • Archives

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:	Show me	Tell me
• Console • Scripting - configure • Scripting - administer		

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:	Tell me:	Guide me (Web servers)
• Generic servers • Custom services	• Generic servers • Templates • Custom services	

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:	Show me	Tell me	Teach 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

- Console
- Scripting

Administer HTTP sessions

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

Documentation: Show me

- Console
- 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: Show me Tell me

- Name server
- Bindings

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 • Scripting	• Cloudscape • 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		Teach me
• Console • Scripting				

Use IBM service integration technologies

		Tell me:		Teach me
		• Overview • Architecture • Mediation		

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 • 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	

- Console
- 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

- Console
- Scripting

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	Teach 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

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.

Documentation Tell me

See also:

- Start, stop, monitor processes
- Other administrative commands
- Custom Java administrative clients (JMX)

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 deployment

Troubleshoot problems that occur either during deployment or shortly afterwards, when you try to access an application that you just deployed for the first time.

Documentation

Troubleshoot administration

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

Documentation

Chapter 3. Setting up the application serving environment

This topic summarizes the contents of the documentation that helps you set up your application serving environment. This information is for administrators, particularly those performing installation, customization, and maintenance of topologies.

“Planning the installation (diagrams)”

In preparation for installation, this topic describes common product topologies that you can install with WebSphere Application Server, Version 6 products.

“Configuring the product after installation” on page 32

This topic describes what to do after installing the product.

You can display the First Steps tool, an easy way to get started with the product.

“Configuring ports” on page 55

This topic provides information about port number settings for Version 6 and previous versions, for use in coexistence and interoperability situations.

“Communicating with Web servers” on page 57

This topic describes how to install and configure WebSphere plug-ins for Web servers, enabling communication between Web servers and application servers.

“Setting up the administrative architecture” on page 79

This topic describes how to configure administrative services.

“Configuring the environment” on page 86

This topic describes how to configure settings for virtual hosts, variables, and shared libraries to assist in handling requests among Web applications, Web containers, and application servers.

“Working with server configuration files” on page 104

This topic describes how to change the default locations of configuration files, as needed. Application server configuration files define the available application servers, their configurations, and their contents.

“Administering application servers” on page 111

This topic describes how to configure individual application servers to provides services for running enterprise applications and their components.

Planning the installation (diagrams)

This topic describes common product topologies that you can install with the product.

Use this topic to understand the capabilities of your product package. Knowing what you can do with the product might influence how you install the product and other installable components on the product disc.

This topic describes topology diagrams and shows you how to create the topologies by showing what components to install for each topology.

Phased installation roadmap

To install a Version 6 production environment, you install the following components:

- The WebSphere Application Server product on your product CD
- A supported Web server, such as the IBM HTTP Server V6 on the product CD
- A binary plug-in module for your Web server from the product CD

You can also use the product CD to install an application client environment on a client machine. Running Java 2 Platform, Enterprise Edition (J2EE) and thin application clients that communicate with WebSphere Application Server requires that elements of the Application Server are installed on the machine on which the client runs. However, if the machine does not have the Application Server installed, you can install Application Server clients to provide a stand-alone client run-time environment for your client applications.

You can use the Rational Web Developer CD in the primary packet of discs to install a fully integrated development environment that includes an exact replica of the Application Server for development testing.

Installation features

Installation features in V6 include:

Feature	Description
The Express product includes Application Server nodes.	You can use the Profile creation wizard to create stand-alone application server nodes after installing the Express product. You do not have to reinstall the product to create additional application servers. All application servers on a machine share the same core product files.
The product CD includes all of the installable components that are required to create an e-business environment.	You can use the product CD to install the IBM HTTP Server, the Web server plug-ins, and the WebSphere Application Server Clients. You do not have to use separate CDs. Separate installation programs exist within component directories on the product CD.
Each installable component has its own installation program.	You can use the V6 launchpad to install any installable component on the product CD. Or you can install each component directly using the **install** command in each component directory.
The launchpad can install any installable product in the primary packet of compact discs.	The launchpad for the Express product can also install the Rational Web Developer on Windows and Linux (Intel) systems. Rational Web Developer is on separate CDs, which requires you to change discs to launch the installation.

Review topology diagrams for each of the following installable components to determine which topology best fits your needs. The diagrams and their accompanying procedures can serve as a roadmap for installing a similar topology.

This topic describes installation scenarios for the following installable components:

- WebSphere Application Server - Express
- Web server plug-ins
- Application clients

In addition to product installation diagrams for the installable components, this topic also links to a roadmap for using the Profile creation wizard, which is new for Version 6. The Profile creation wizard lets you create run-time environments for application server processes.

Each of the following installation scenarios includes topology diagrams and associated installation steps. Each step links to a specific procedure for installing a component or to a description of a command or tool.

1. Review the installation scenarios for the Express product, as described in “Planning to install WebSphere Application Server - Express” on page 17.
2. Review the installation scenarios for the WebSphere Application Server plug-ins, as described in “Planning to install Web server plug-ins” on page 22.
3. Review the installation scenarios for the application clients, as described in “Planning to install WebSphere Application Server Clients” on page 25.
4. Review the installation scenarios for the Profile creation wizard, as described in “Planning to create application server environments” on page 26.
5. **Optional:** Review interoperability and coexistence diagrams to know what is possible with Version 6.

WebSphere Application Server V6 can interoperate with your other e-business systems, including other versions of WebSphere Application Server. *Interoperability* provides a communication mechanism for WebSphere Application Server nodes that are at different versions. *Coexistence* describes multiple versions or instances running on the same machine, at the same time.

Interoperability support enhances migration scenarios with more configuration options. It often is convenient or practical to interoperate during the migration of a configuration from an earlier WebSphere Application Server version to a later one when some machines are at the earlier version and some machines are at the later version. The mixed environment of machines and application components at different software version levels requires interoperability and coexistence.

It is often impractical, or even physically impossible, to migrate all the machines and applications within an enterprise at the same time. Understanding multiversion interoperability and coexistence is therefore an essential part of a migration between version levels.

See the following topics for more information:

a. Interoperating

b. Setting up Version 4.0.x and Version 6 coexistence

c. Setting up Version 5 and Version 6 coexistence

d. Setting up Version 6 coexistence

6. **Optional:** Consider performance when designing your network, as described in "Queuing network" on page 27.

You can review installation scenarios to identify the specific steps to follow when installing more than one component on a single machine or on separate machines.

After determining an appropriate installation scenario, you are ready to install the necessary components and to configure the products for the system that you selected.

Planning to install WebSphere Application Server - Express

This topic describes common installation scenarios and links to component installation procedures for each scenario.

IBM WebSphere Application Server - Express, Version 6 is an integrated platform that contains an Application Server, Web development tools, a Web server, and additional supporting software and documentation.

The following information describes scenarios for installing the product in various topologies on one or more machines:

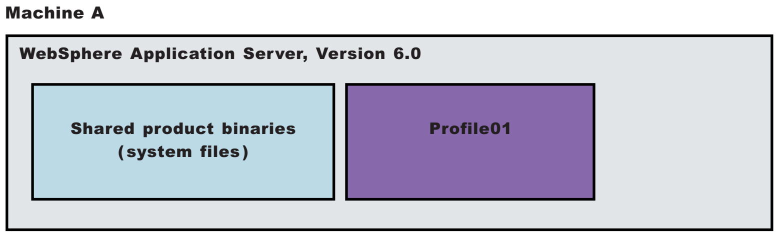
- **Scenario 1:** Single-machine installation of WebSphere Application Server - Express
- **Scenario 2:** Single-machine installation of WebSphere Application Server - Express and a Web server
- **Scenario 3:** Two-machine installation of WebSphere Application Server - Express and a Web server
- **Scenario 4:** Creating multiple profiles that use one installation of WebSphere Application Server - Express
- **Scenario 5:** Single-machine installation of Rational Web Developer
- **Scenario 6:** Multiple-machine installation of WebSphere Application Server - Express, Rational Web Developer, and a Web server

Each scenario includes a diagram and a list of detailed installation steps.

- **Scenario 1:** Install WebSphere Application Server - Express on a single machine.

  Installing WebSphere Application Server - Express by itself on a single machine creates a stand-alone application server, which is server1. Installing Express creates a set of system files and a *profile* for the application server. The profile is a separate set of files that define the application server environment.

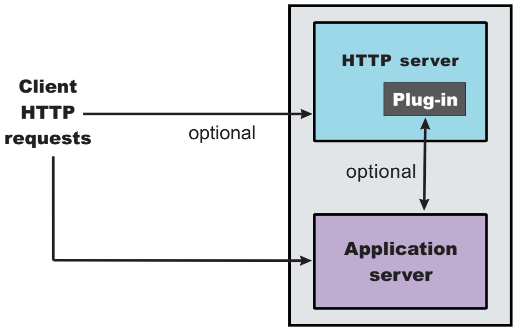
In this scenario, the application server uses its internal HTTP transport chain for communication, which is suitable for handling an application with a relatively low request work load. For example, this type of installation can support a simple test environment or a departmental intranet environment.


Machine A
WebSphere Application Server, Version 6.0
Shared product binaries (system files)
Profile01

1. Install WebSphere Application Server - Express. See the *Installing your application serving environment* PDF.

- **Scenario 2:** Install WebSphere Application Server - Express and a Web server on a single machine.

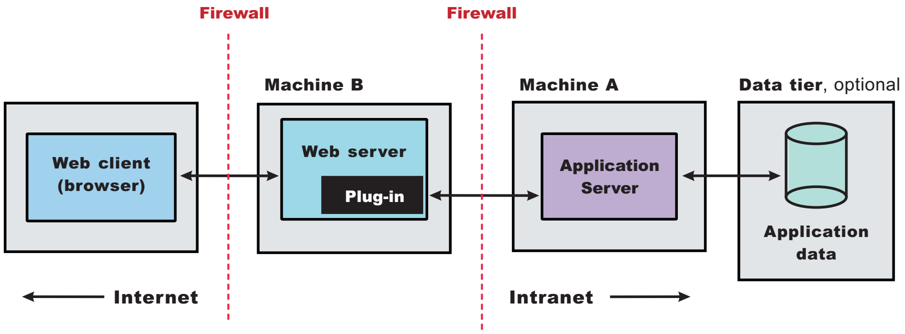
  Installing a Web server, such as IBM HTTP Server, on the same machine as the application server provides a more robust Web server environment. Installing a Web server plug-in is a requirement for the Web server to communicate with the application server. This type of installation supports rigorous testing environments or production environments that do not require a firewall. However, this is not a typical production environment.



1. Install WebSphere Application Server - Express See the *Installing your application serving environment* PDF.
2. Install IBM HTTP Server or another supported Web server. See your Web server documentation.
3. Install the Web server plug-ins and configure the Web server using the Plug-ins installation wizard. See the *Installing your application serving environment* PDF.

- **Scenario 3:** Install WebSphere Application Server - Express and a Web server on separate machines.

  In the typical production environment, the application server on one machine communicates with a Web server on a separate (remote) machine through the Web server plug-in. Optional firewalls can provide additional security for the application server machine.



1. Install WebSphere Application Server - Express on Machine A. See the *Installing your application serving environment* PDF.
2. Install IBM HTTP Server or another supported Web server on Machine B. See your Web server documentation.
3. Install the Web server plug-ins and configure the Web server using the Plug-ins installation wizard on Machine B. See the *Installing your application serving environment* PDF.
4. The Plug-ins installation wizard creates a script named `configure`*`Web_server_name`* in the *plugins_install_root*/`bin` directory on Machine B. Copy the script to the *install_root*/`bin` directory on Machine A.
5. Run the `configure`*`Web_server_name`* script to create a Web server definition in the administrative console. You can then use the administrative console to manage the Web server.
6. Propagate the `plugin-cfg.xml` file from the application server to the Web server using the administrative console. Click **Servers > Web server > Propagate Plug-in**. (Web servers other than IBM HTTP Server require manual propagation.)

- **Scenario 4:** Install multiple stand-alone application servers on one machine and a Web server on a separate machine.

  A profile is a separate data partition containing the files that define the run-time environment for an application server. A default profile is created during the installation of the Express product. Create additional profiles using the Profile creation wizard. Each profile defines a separate stand-alone application server that has its own administrative interface.

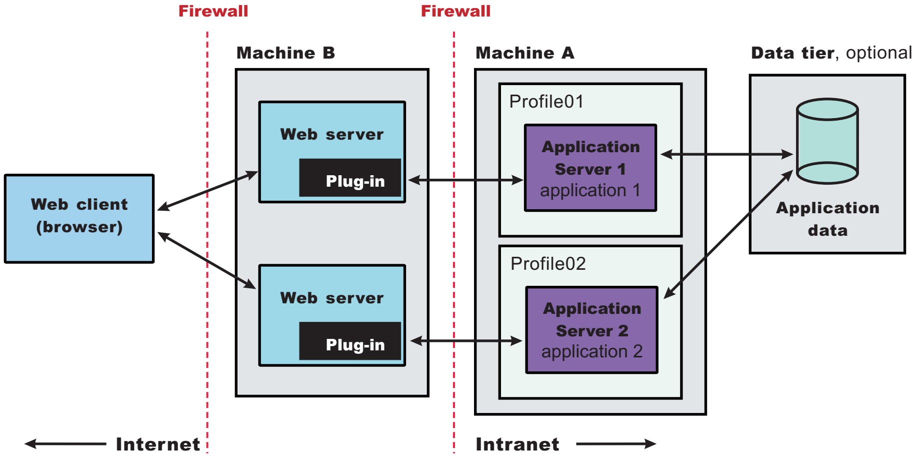
  After creating a profile and installing a dedicated Web server, use the Plug-ins installation wizard to install a Web server plug-in and to update the Web server configuration file. The Web server can then communicate with the application server.

  This topology lets each profile have unique applications, configuration settings, data, and log files, while sharing the same set of core product files. Creating multiple profiles creates multiple application server environments that you can dedicate to different purposes.

  For example, each application server on a Web site can serve a different application. In another example, each application server can be a separate test environment that you assign to a programmer or a development team.

  Updating the core product files

  Another feature of having multiple profiles is enhanced serviceability. When a refresh pack or fix pack updates the core product files on a machine, all of the application server profiles that were created from the core product files begin using the updated files. In some situations, you might prefer to not update all of the application servers on a machine. In such situations, simply install the product a second time to create a second set of core product files. Create application server profiles from both installations to manage the product updates incrementally.

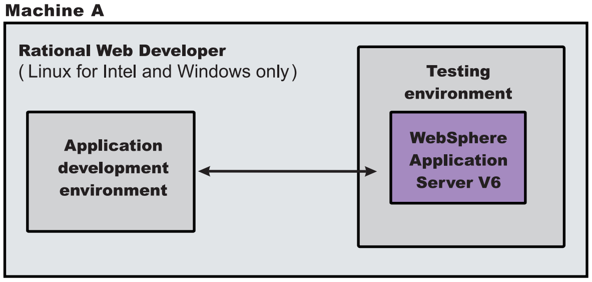


1. Install WebSphere Application Server - Express on Machine A. See the *Installing your application serving environment* PDF.
2. Install IBM HTTP Server or another supported Web server on Machine B. See your Web server documentation.
3. Install the Web server plug-ins and configure the Web server using the Plug-ins installation wizard on Machine B. See the *Installing your application serving environment* PDF.
4. The Plug-ins installation wizard creates a script named `configure`*`Web_server_name`* in the *plugins_install_root*/`bin` directory on Machine B. Copy the script to the *install_root*/`bin` directory on Machine A.
5. Run the `configure`*`Web_server_name`* script to create a Web server definition in the administrative console. You can then use the administrative console to manage the Web server.
6. Propagate the `plugin-cfg.xml` file from the application server to the Web server using the administrative console. Click **Servers > Web server > Propagate Plug-in**. (Web servers other than IBM HTTP Server require manual propagation.)
7. Create the second Application Server profile using the Profile creation wizard on Machine A. Make the profile the default profile during the profile creation by selecting the check box on the appropriate panel.

   The script that the Plug-ins installation wizard creates works on the default profile only. So, this script can only create a Web server definition on the profile that is the default profile at the time that the script runs.
8. Install a second IBM HTTP Server or another supported Web server on Machine B. See your Web server documentation.
9. On Machine B, install the Web server plug-ins to configure the second Web server using the Plug-ins installation wizard. Both Web servers share a single installation of the plug-in binaries but must be configured individually. See the *Installing your application serving environment* PDF.
10. The Plug-ins installation wizard creates a script named `configure`*`Web_server_name`* for the second Web server. The script is in the *plugins_install_root*/`bin` directory on Machine B. Copy the script to the *install_root*/`bin` directory on Machine A.
11. Run the `configure`*`Web_server_name`* script to create a Web server definition in the administrative console. You can then use the administrative console to manage the Web server.
12. Propagate the `plugin-cfg.xml` file from the second application server to the Web server using the administrative console. Click **Servers > Web server > Propagate Plug-in**. (Web servers other than IBM HTTP Server require manual propagation.)

- **Scenario 5:** Install Rational Web Developer on a single machine.

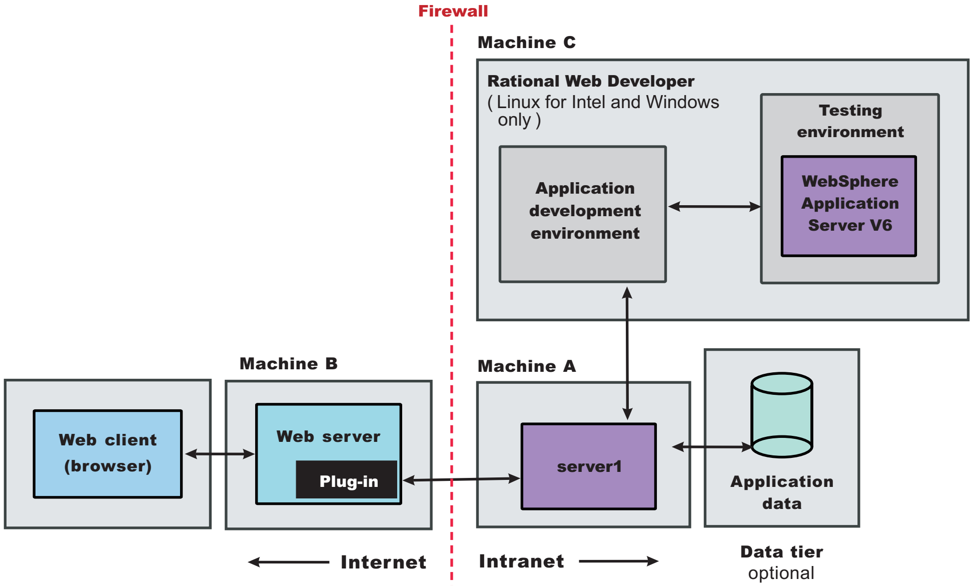
  Install Rational Web Developer by itself to create an integrated development environment. Developers can test applications in the test environment for V6 that is included in the Rational Web Developer product.


Machine A
Rational Web Developer
(Linux for Intel and Windows only)
Application development environment
Testing environment
WebSphere Application Server V6

1. Install Rational Web Developer.

- **Scenario 6:** Install WebSphere Application Server - Express, Rational Web Developer, and a Web server on multiple machines.

  In this integrated testing environment, install the application server behind an optional firewall on one machine and the Web server on a separate (remote) machine. The Web server plug-in lets the application server and Web server communicate with each other. Install Rational Web Developer on a separate Linux (Intel) or Windows machine. This type of installation is useful for quality assurance because the test environment closely mirrors the typical production environment.



1. Install WebSphere Application Server - Express on Machine A. See the *Installing your application serving environment* PDF.
2. Install IBM HTTP Server or another supported Web server on Machine B. See your Web server documentation.
3. Install the Web server plug-ins and configure the Web server using the Plug-ins installation wizard on Machine B. See the *Installing your application serving environment* PDF.
4. The Plug-ins installation wizard creates a script named `configure`*`Web_server_name`* in the *plugins_install_root*/`bin` directory on Machine B. Copy the script to the *install_root*/`bin` directory on Machine A.
5. Run the `configure`*`Web_server_name`* script to create a Web server definition in the administrative console. You can then use the administrative console to manage the Web server.
6. Propagate the `plugin-cfg.xml` file from the application server to the Web server using the administrative console. Click **Servers > Web server > Propagate Plug-in**. (Web servers other than IBM HTTP Server require manual propagation.)
7. Install Rational Web Developer on Machine C.

You can review common installation scenarios to find a possible match for the topology that you intend to install. Each product installation diagram provides a high-level procedure for installing the components that comprise the topology.

After determining a possible topology, you are ready to follow the detailed installation instructions for each product that you plan to install.

Planning to install Web server plug-ins

This topic describes common installation scenarios and links to component installation procedures for each scenario.

The primary production configuration is an application server on one machine and a Web server on a separate machine. This configuration is referred to as a *remote* configuration. Contrast the remote configuration to the local configuration, where the application server and the Web server are on the same machine.

The Plug-ins installation wizard has four main tasks:

- Installs the binary plug-in module on the Web server machine.
- Configures the Web server configuration file on the Web server machine to point to the binary plug-in module and to the XML configuration file for the binary module.
- Installs a temporary XML configuration file for the binary module (`plugin-cfg.xml`) on the Web server machine in remote scenarios.
- Creates the configuration for a Web server definition on the application server machine. The wizard processes the creation of the Web server definition differently depending on the scenario:
  - Recommended remote stand-alone Application Server installation:

    Creates a configuration script that you run on the application server machine. Install the Web server and its plug-in on a different machine than the application server. This configuration is recommended for a production environment.
  - Local stand-alone Application Server installation:

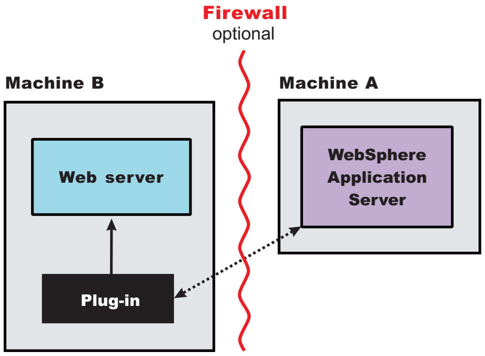
    Detects the default profile on a local application server machine and creates the Web server definition for it directly. Install the Web server and its plug-in on the same machine with the application server. This configuration is for development and test environments.

Select a link to go to the appropriate steps in the following procedure.

- **Set up a remote Web server installation.**

  The remote Web server configuration is recommended for production environments.

  The remote installation installs the Web server plug-in on the Web server machine when the application server is on a separate machine, such as shown in the following graphic:



Remote installation scenario

Table 1. Installation and configuration

Step	Machine	Task
1	A	Install your WebSphere Application Server product. See ″Installing the product and additional software″ in the information center.
2	B	Install IBM HTTP Server or another supported Web server. See ″Installing IBM HTTP Server″ in the information center.
3	B	Install the binary plug-in module using the Plug-ins installation wizard. See ″Configuring a Web server and an application server on separate machines (remote)″ in the information center. The script for creating and configuring the Web server is created under the *`plug-ins_install_root`*`/ bin` directory.
4	B	Copy the `configure`*`Web_server_name`* script to Machine A. If one machine is running under Linux or UNIX and the other machine is running under Windows, copy the script from the *`plug-ins_install_root`*`/ bin/ crossPlatformScripts` directory.
5	A	Paste the `configure`*`Web_server_name`* script from Machine B to the *`was_install_root`*`/ bin` directory on Machine A.
6	A	Run the script from a command line.
7	A	Verify that the application server is running. Open the administrative console and save the changed configuration.
8	B	Linux UNIX Run the *plug-ins_install_root*`/setupPluginCfg.sh` script for a Domino Web Server before starting a Domino Web server.Otherwise, start the Web server.
9	B	Run the snoop servlet. See ″Troubleshooting installation″ in the information center. To verify with your own application, regenerate and propagate the `plugin-cfg.xml` file after installing the application.

Regeneration of the `plugin-cfg.xml` file

During the installation of the plug-ins, the temporary `plugin-cfg.xml` file is installed on Machine B in the *`plug-ins_install_root`*`/ config/ web_server_name` directory.

The Web server plug-in configuration service regenerates the `plugin-cfg.xml` file automatically.

To use the real `plugin-cfg.xml` file from the application server, propagate the `plugin-cfg.xml` file as described in the next section.

Propagation of the `plugin-cfg.xml` file

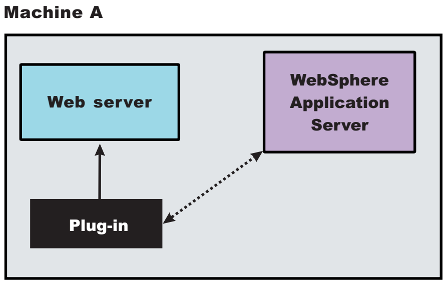
The Web server plug-in configuration service propagates the `plugin-cfg.xml` file automatically for IBM HTTP Server 6.0 only.

For all other Web servers, propagate the plug-in configuration file manually. Copy the `plugin-cfg.xml` file from the *`profiles_install_root`*`/ config/ cells/` *`cell_name`*`/ nodes/` *`Web_server_name`*`_node/ servers/` *`web_server_name`* directory on Machine A. Paste the file into the *`plug-ins_install_root`*`/ config/` *`web_server_name`* directory on Machine B.

- **Set up a local Web server configuration.**

  The local Web server configuration is recommended for a development or test environment.

  A local installation includes the Web server plug-in, the Web server, and the application server on the same machine:



Local installation scenario

Table 2. Installation and configuration

Step	Machine	Task
1	A	Install your WebSphere Application Server product. See ″Installing the product and additional software″ in the information center.
2	A	Install IBM HTTP Server or another supported Web server. See ″Installing IBM HTTP Server″ in the information center.
3	A	Install the binary plug-in module using the Plug-ins installation wizard. See ″Configuring a Web server and an application server on separate machines (remote)″ in the information center. The Web server definition is automatically created and configured during the installation of the plug-ins.
4	A	Verify that the application server is running. Open the administrative console and save the changed configuration.
5	B	Linux UNIX Run the *plug-ins_install_root*/`setupPluginCfg.sh` script for a Domino Web Server before starting a Domino Web server.Otherwise, start the Web server.
6	B	Run the snoop servlet. To verify with your own application, regenerate and propagate the `plugin-cfg.xml` file after installing the application.

Regeneration of the `plugin-cfg.xml` file

The Web server plug-in configuration service regenerates the `plugin-cfg.xml` file automatically.

The `plugin-cfg.xml` file is generated in the *profiles_install_root*/ *profile_name*/ config/ cells/ *cell_name*/ nodes/ *Web_server_name*_node/ servers/ *web_server_name* directory. The generation occurs when the Web server definition is created.

Propagation of the plugin-cfg.xml file

The local file does not require propagation.

You can set up a remote or local Web server by installing Application Server, the Web server, and then the Web server plug-ins.

See Web server configuration for more information about the files involved in configuring a Web server.

See ″Editing Web server configuration files″ in the information center for details for information about the logic behind the processing scenarios for the Plug-ins installation wizard.

See Editing Web server configuration files for information about how the Plug-ins installation wizard configures supported Web servers.

See Installing Web server plug-ins for information about other installation scenarios for installing Web server plug-ins.

Planning to install WebSphere Application Server Clients

This topic helps you examine typical topologies and uses for WebSphere Application Server Clients.

This topic is one in a series of topics described in "Planning the installation (diagrams)" on page 15. Consider all of the planning scenarios that are mentioned in the parent article to determine the best approach to installing your e-business network. This topic describes installing and using the WebSphere Application Server Clients.

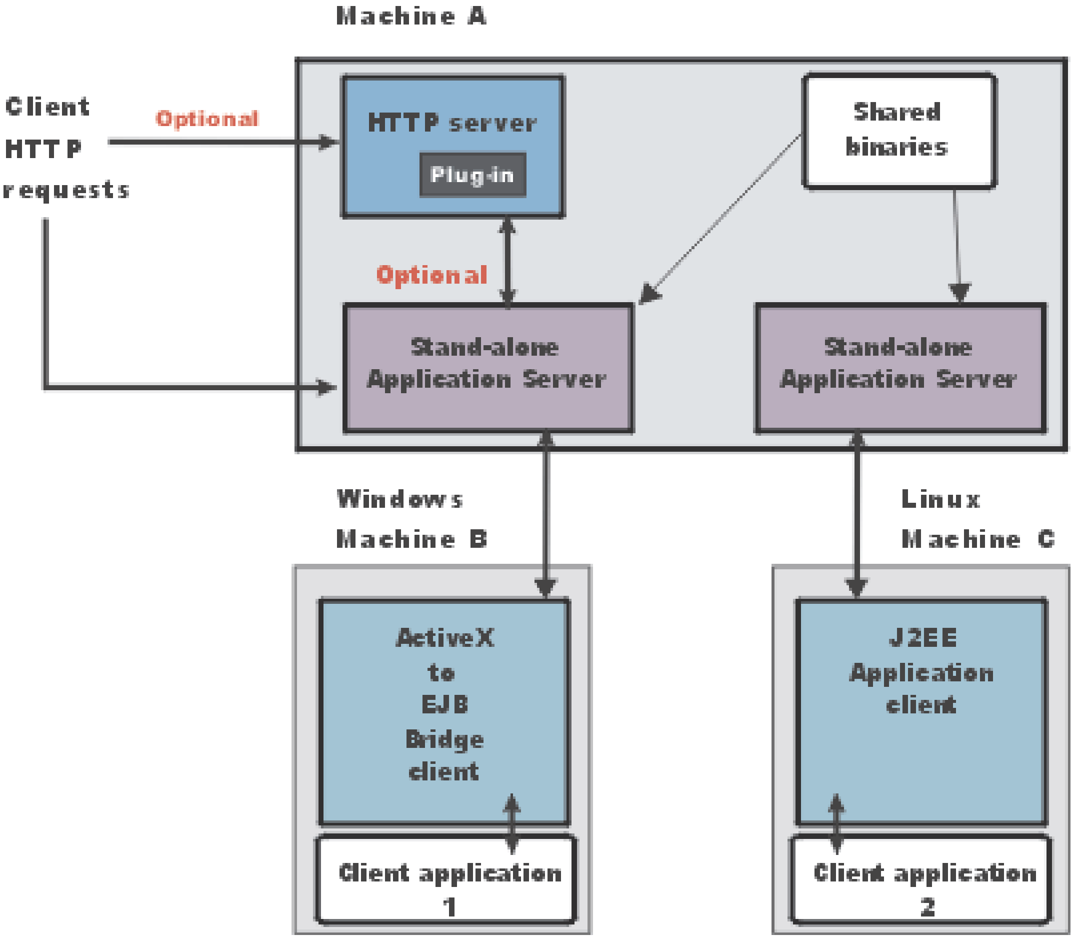
In a traditional client server environment, the client requests a service and the server fulfills the request. Multiple clients use a single server. Clients can also access several different servers. This model persists for Java clients except that now these requests use a client run-time environment.

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 now consists of the following models:

- ActiveX application client
- Applet client
- J2EE application client
- Pluggable and thin application clients

The following graphic shows a topology for installing the Application Client and using client applications:



The example shows two types of application clients installed in a topology that uses client applications to access applications and data on Machine A:

- The ActiveX application client on Machine B is a Windows only client that uses the Java Native Interface (JNI) architecture to programmatically access the Java virtual machine (JVM) API. 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.
- The J2EE application client on Machine C is a Java application program that accesses enterprise beans, Java Database Connectivity (JDBC) APIs, and Java Message Service message queues. The application program must configure the execution environment of the J2EE application client and use the Java Naming and Directory Interface (JNDI) name space to access resources.

Use the following procedure as a roadmap for installing the Application Client.

1. Install the WebSphere Application Server product from your product CD on Machine A to establish the core product files.
2. Use the Profile creation wizard to create the additional stand-alone application server profile.
3. Use the administrative console of each application server to deploy any user applications.
4. Use the administrative console of each application server to create a Web server configuration for the Web server.
5. Use the administrative console of each application server to regenerate each `plugin-cfg.xml` file in the local Web server configuration.
6. Install the IBM HTTP Server from the product CD on Machine A.
7. Use the Plug-ins installation wizard to install the plug-in for IBM HTTP Server on Machine A.

   The wizard automatically configures the HTTP Server to communicate with the first application server.
8. Install the Application Client from your product CD on Machine B.
   a. Select the Custom install type.
   b. Select the ActiveX to EJB Bridge feature.
   c. Select to add the Java run time to the system path.
   d. Select the Java run time as the default JRE, which adds the Java run time path to the beginning of the system path.
9. Install the Application Client from your product CD on Machine C.
   a. Select the Custom install type.
   b. Select the J2EE application client feature.

This topic can help you plan run-time environments for client applications.

See “Application Client for WebSphere Application Server” on page 839 for information about creating client applications.

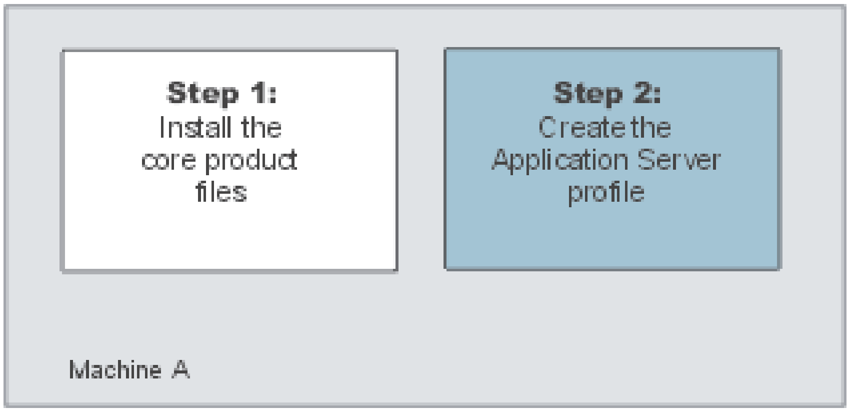
Planning to create application server environments

Application server profiles are the run-time environments for application server processes. This topic describes common scenarios for creating application server profiles and provides links to profile creation procedures for each scenario.

Install the core product files for a WebSphere Application Server product before using the Profile creation wizard to create additional application server run-time environments.

This topic describes how to use the Profile creation wizard to create application server profiles. Each profile is a run-time environment for the application server that it defines, with data files, configuration files, applications, and an administrative console.

Create a stand-alone application server, as described in “Using the Profile creation wizard to create an application server” on page 38.



The installation procedure creates an application server during installation named server1. However, you can use the Profile creation wizard to create more stand-alone application servers on a machine where server1 or another application server already exists.

You can create additional application server processes using the Profile creation wizard.

After installing the product and optionally adding more application servers with the Profile creation wizard, you are ready to deploy applications to test the environment.

Queuing network

WebSphere Application Server contains interrelated components that must be harmoniously tuned to support the custom needs of your end-to-end e-business application. These adjustments help the system achieve maximum throughput while maintaining the overall stability of the system. This group of interconnected components is known as a queuing network. These queues or components include the network, Web server, Web container, EJB container, data source, and possibly a connection manager to a custom back-end system. Each of these resources represents a queue of requests waiting to use that resource. Various queue settings include:

- IBM HTTP Server: MaxClients for UNIX and ThreadsPerChild for Windows NT and Windows 2000 systems described in “Web server tuning parameters” on page 73.
- Web container: **Maximum size** described in “Thread pool settings” on page 120, **MaxKeepAliveConnections** and **MaxKeepAliveRequests** described in “HTTP transport custom properties” on page 132.
- **Tuning Object Request Brokers** explained in “Tuning application servers” on page 167.
- Data source **connection pooling** and **statement cache size** are explained in the *Developing and deploying applications* PDF.

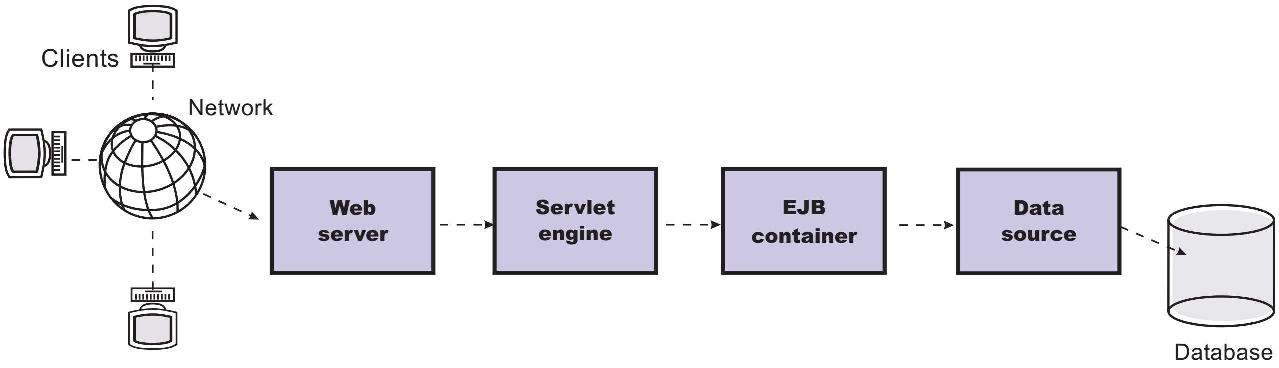


Figure Reference 1: WebSphere queuing network

Most of the queues that make up the queuing network are closed queues. A closed queue places a limit on the maximum number of requests present in the queue, while an open queue has no limit. A closed queue supports tight management of system resources. For example, the Web container thread pool setting controls the size of the Web container queue. If the average servlet running in a Web container creates 10MB of objects during each request, a value of 100 for thread pools limits the memory consumed by the Web container to 1GB.

In a closed queue, requests can be active or waiting. An active request is doing work or waiting for a response from a downstream queue. For example, an active request in the Web server is doing work, such as retrieving static HTML, or waiting for a request to complete in the Web container. A waiting request is waiting to become active. The request remains in the waiting state until one of the active requests leaves the queue.

All Web servers supported by WebSphere Application Server are closed queues, as are WebSphere Application Server data sources. You can configure Web containers as open or closed queues. In general, it is best to make them closed queues. EJB containers are open queues. If there are no threads available in the pool, a new one is created for the duration of the request.

If enterprise beans are called by servlets, the Web container limits the number of total concurrent requests into an EJB container, because the Web container also has a limit. The Web container limits the number of total concurrent requests only if enterprise beans are called from the servlet thread of execution. Nothing prevents you from creating threads and bombarding the EJB container with requests. Therefore, servlets should not create their own work threads.

Queuing and clustering

Cloning application servers can be a valuable asset in configuring highly-scalable production environments, especially when the application is experiencing bottlenecks that are preventing full CPU utilization of symmetric multiprocessing (SMP) servers. When adjusting the WebSphere Application Server system queues in clustered configurations, remember that when a server is added to a cluster, the server downstream receives twice the load.

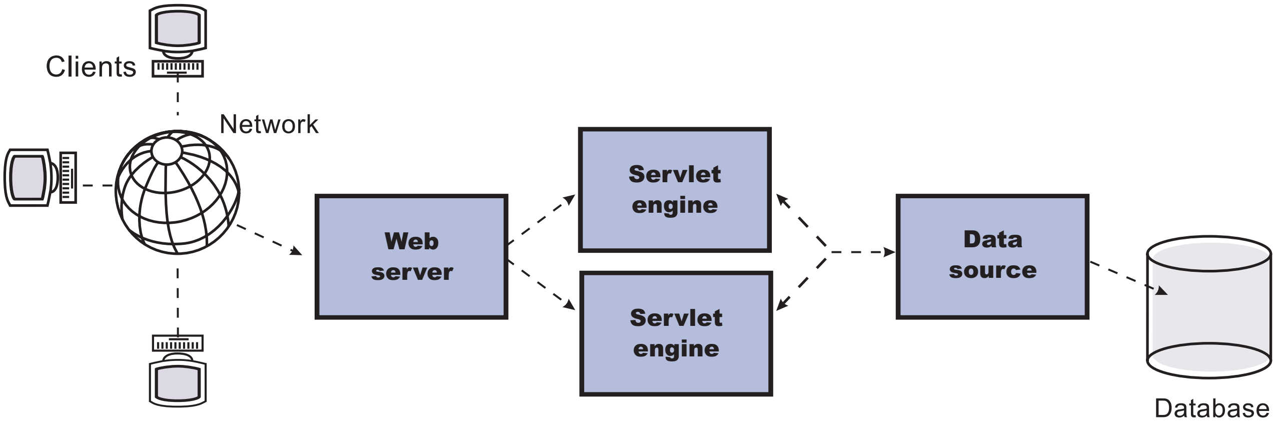


Figure Reference 1: Clustering and queuing

Two servlet engines are located between a Web server and a data source. It is assumed that the Web server, servlet engines and data source, but not the database, are all running on a single SMP server. Given these constraints, the following queue considerations must be made:

- Double the Web server queue settings to ensure ample work is distributed to each Web container.
- Reduce the Web container thread pools to avoid saturating a system resource like CPU or another resource that the servlets are using.
- Reduce the data source to avoid saturating the database server.
- Reduce Java heap parameters for each instance of the application server. For versions of the Java virtual machine (JVM) shipped with WebSphere Application Server, it is crucial that the heap from all JVMs remain in physical memory. For example, if a cluster of four JVMs is running on a system, enough physical memory must be available for all four heaps.

Queue configuration tips

The following section outlines a methodology for configuring the WebSphere Application Server queues. Moving the database server onto another machine or providing more powerful resources, for example a faster set of CPUs with more memory, can dramatically change the dynamics of your system.

There are four tips for queuing:

- **Minimize the number of requests in WebSphere Application Server queues**.

  In general, requests wait in the network in front of the Web server, rather than waiting in WebSphere Application Server. This configuration only supports those requests that are ready for processing to enter the queuing network. Specify that the queues furthest upstream or closest to the client are slightly larger, and queues further downstream or furthest from the client are progressively smaller.

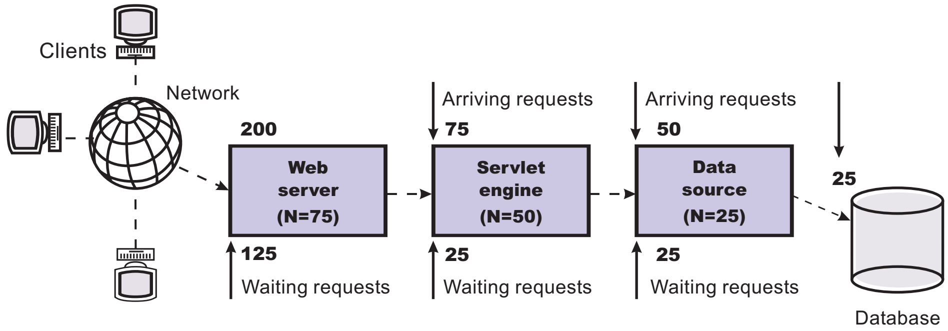
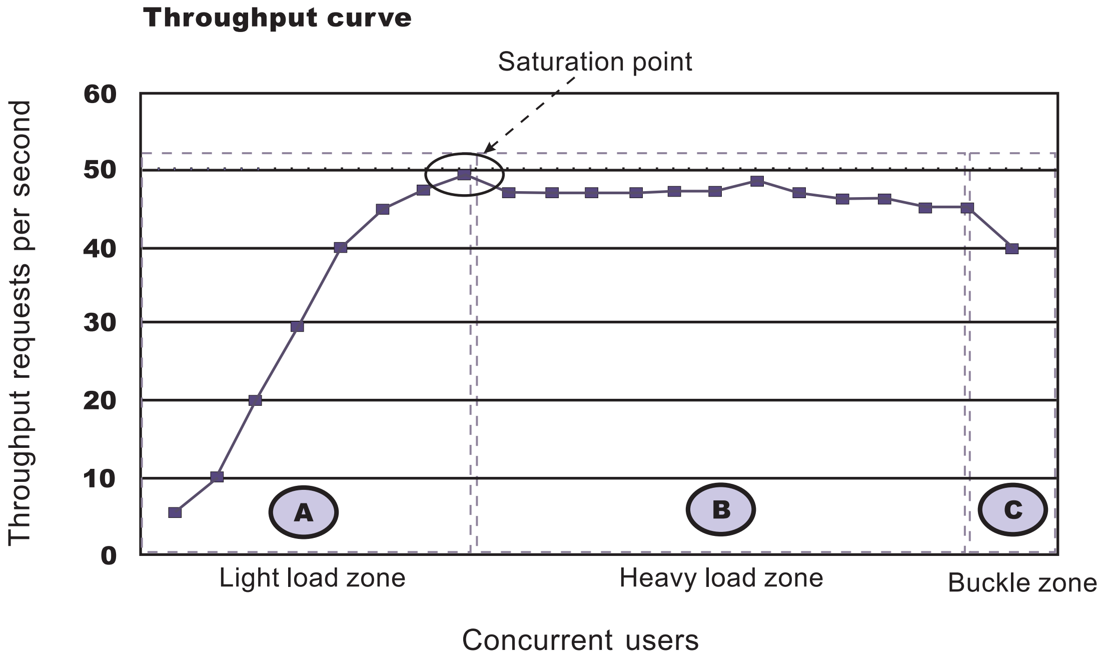


Figure Reference 1: Upstream queuing network

Queues in the queuing network become progressively smaller as work flows downstream. When 200 client requests arrive at the Web server, 125 requests remain queued in the network because the Web server is set to handle 75 concurrent clients. As the 75 requests pass from the Web server to the Web container, 25 requests remain queued in the Web server and the remaining 50 are handled by the Web container. This process progresses through the data source until 25 user requests arrive at the final destination, the database server. Because there is work waiting to enter a component at each point upstream, no component in this system must wait for work to arrive. The bulk of the requests wait in the network, outside of WebSphere Application Server. This type of configuration adds stability, because no component is overloaded.

- **Draw throughput curves to determine when the system capabilities are maximized**.

  You can use a test case that represents the full spirit of the production application by either exercising all meaningful code paths or using the production application. Run a set of experiments to determine when the system capabilities are fully stressed or when it has reached the saturation point. Conduct these tests after most of the bottlenecks are removed from the application. The goal of these tests is to drive CPUs to near 100% utilization. For maximum concurrency through the system, start the initial baseline experiment with large queues. For example, start the first experiment with a queue size of 100 at each of the servers in the queuing network: Web server, Web container and data source. Begin a series of experiments to plot a throughput curve, increasing the concurrent user load after each experiment. For example, perform experiments with one user, two users, five, 10, 25, 50, 100, 150 and 200 users. After each run, record the throughput requests per second, and response times in seconds per request. The curve resulting from the baseline experiments resembles the following typical throughput curve shown as follows:



The WebSphere Application Server throughput is a function of the number of concurrent requests present in the total system. Section A, the light load zone, shows that the number of concurrent user requests increases, the throughput increases almost linearly with the number of requests. At light loads, concurrent requests face very little congestion within the WebSphere Application Server system queues. At some point, congestion starts to develop and throughput increases at a much lower rate until it reaches a saturation point that represents the maximum throughput value, as determined by some bottleneck in the WebSphere Application Server system. The most manageable type of bottleneck occurs when the WebSphere Application Server machine CPUs become fully utilized because adding CPUs or more powerful CPUs fixes the bottleneck.

In the heavy load zone or Section B, as the concurrent client load increases, throughput remains relatively constant. However, the response time increases proportionally to the user load. That is, if the user load is doubled in the heavy load zone, the response time doubles. At some point, represented by Section C, the buckle zone, one of the system components becomes exhausted. At this point, throughput starts to degrade. For example, the system might enter the buckle zone when the network connections at the Web server exhaust the limits of the network adapter or if the requests exceed operating system limits for file handles.

If the saturation point is reached by driving CPU utilization close to 100%, you can move on to the next step. If the saturation CPU occurs before system utilization reaches 100%, it is likely that another bottleneck is being aggravated by the application. For example, the application might be creating Java objects causing excessive garbage collection bottlenecks in the Java code.

There are two ways to manage application bottlenecks: remove the bottleneck or clone the bottleneck. The best way to manage a bottleneck is to remove it. You can use a Java-based application profiler, such as Rational Application Developer, Performance Trace Data Visualizer (PTDV), Borland's Optimizeit, JProbe or Jinsight to examine overall object utilization.

- **Decrease queue sizes while moving downstream from the client**.

  The number of concurrent users at the throughput saturation point represents the maximum concurrency of the application. For example, if the application saturates WebSphere Application Server at 50 users, using 48 users might produce the best combination of throughput and response time. This value is called the Max Application Concurrency value. Max Application Concurrency becomes the preferred value for adjusting the WebSphere Application Server system queues. Remember, it is desirable for most users to wait in the network; therefore, queue sizes should increase when moving downstream farther from the client. For example, given a Max Application Concurrency value of 48, start with system queues at the following values: Web server 75, Web container 50, data source 45. Perform a set of additional experiments adjusting these values slightly higher and lower to find the best settings.

To help determine the number of concurrent users, view the Servlet Engine Thread Pool and Concurrently Active Threads metric in the Tivoli Performance Viewer.

- **Adjust queue settings to correspond to access patterns**.

  In many cases, only a fraction of the requests passing through one queue enters the next queue downstream. In a site with many static pages, a number of requests are fulfilled at the Web server and are not passed to the Web container. In this circumstance, the Web server queue can be significantly larger than the Web container queue. In the previous example, the Web server queue was set to 75, rather than closer to the value of Max Application Concurrency. You can make similar adjustments when different components have different execution times.

  For example, in an application that spends 90% of its time in a complex servlet and only 10% of its time making a short JDBC query, on average 10% of the servlets are using database connections at any time, so the database connection queue can be significantly smaller than the Web container queue. Conversely, if the majority of servlet execution time is spent making a complex query to a database, consider increasing the queue values at both the Web container and the data source. Always monitor the CPU and memory utilization for both the WebSphere Application Server and the database servers to verify that the CPU or memory are not saturating.

Configuring the product after installation

This topic summarizes how to configure the application serving environment.

Use the First steps console to configure and test the WebSphere Application Server environment after installation.

This procedure uses the First steps console to launch the installation verification test (IVT) that tests and verifies your WebSphere Application Server - Express environment. This procedure also uses the First steps console to launch the Profile creation wizard to create an additional Application Server.

1. Start the First steps console by selecting the check box on the last panel of the wizard.

   The First steps console can start automatically at the end of the installation. Select the check box on the last panel of the Installation wizard.

   The First steps console is an easy way to start using the product. The console provides one-stop access to the administrative console, Samples Gallery, Profile creation wizard, installation verification test, Migration wizard, and other activities.

   See the description of the “firststeps command” on page 33 for more information.
2. Click **Installation verification** on the First steps console.

   The installation verification test starts the Application Server process named server1 and runs several tests to verify that the server1 process can start without errors.

   See “Using the installation verification test” on page 53 for more information.
3. Click **Profile creation wizard** on the First steps console to create an Application Server profile.

   You can create multiple Application Servers on your system without installing the product again.

   See “Using the Profile creation wizard to create an application server” on page 38.
4. Start the First steps console by selecting the check box on the last panel of the Profile creation wizard.

   This First steps console belongs to the Application Server profile that you just created. Each profile has its own First steps console.
5. Click **Installation verification** on the First steps console.

   The installation verification test starts the new Application Server process named server1 and runs several tests to verify that the server1 process can start without error.

This procedure results in configuring and testing the Application Server environment.

See “Planning to install WebSphere Application Server - Express” on page 17 for diagrams of topologies that you can create using the First steps console and the Profile creation wizard.

firststeps command

The **firststeps** command starts the First steps console.

The First steps console

The First steps console is a post-installation ease-of-use tool for directing WebSphere Application Server elements from one place. Options display dynamically on the First steps console, depending on features you install. With all of the options present, you can use the First steps console to start or stop the application server, verify the installation, access the information center, access the administrative console, launch the Migration wizard, or access the Samples gallery.

Select the check box to start the First steps console at the end of the product installation.

You can also start the First steps console from the command line as described later.

Installation verification

This option starts the installation verification test (IVT). The test consists of starting and monitoring the application server during its start up.

If this is the first time that you have used the First steps console since creating an application server profile, click **Installation verification** to verify that all is well with your installation. The verification process starts the application server.

If you select the **Installation verification** option, the **Start the server** option is grayed out while the IVT is running.

The IVT provides the following useful information about the application server:

- The server name: server1
- The name of the profile
- The profile file path
- The type of profile: default
- The cell name
- The node name
- The current encoding
- The port number for the administrative console
- Various informational messages that include the location of the `SystemOut.log` file and how many errors are listed within the file
- A completion message

Start the server

This option toggles to **Stop the server** when the application server is running.

After selecting the **Start the server** option, an output screen displays with status messages. The success message informs you that the server is open for e-business. Then the menu item changes to **Stop the server**.

If you select the **Start the server** option, the **Installation verification** option is grayed out while the application server is running.

Administrative console

This option is grayed out until the application server is running.

The administrative console is a configuration editor that runs in a Web browser. The administrative console lets you work with XML configuration files for the application server. To launch the administrative console, click **Administrative console**. You can also point your browser to http://localhost:9060/ibm/console to start the administrative console. Substitute your own host name in the address if the localhost variable does not resolve correctly. As the administrative console opens, it prompts you for a login name. This is not a security item, but merely a tag to identify configuration changes that you make during the session. Secure signon is also available.

Profile creation wizard

This option starts the Profile creation wizard. The wizard lets you create additional application servers. A *profile* consists of files that define the run-time environment for the application server. Each environment has its own administrative interface. This means that the new application server has its own administrative console.

Each application server has its own First steps console. The location of the command is within the set of files in the profile. A prompt to launch the First steps console displays on the last panel of the Profile creation wizard.

Samples gallery

This option starts the Samples gallery. The option is grayed out until you start the application server. The option displays when you have installed the Samples during installation. The typical installation includes the Samples by default.

From the First steps console, click **Samples gallery** to explore the application Samples. Alternatively you can point your browser directly to http://localhost:9080/WSsamples. Substitute your own host name in the address if the localhost variable does not resolve correctly. The Web address is case sensitive. Substitute your own host name in the address.

Information center for WebSphere Application Server

This option links you to the online information center at the http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp IBM Web address.

Migration wizard

This option starts the Migration wizard. The Migration wizard is a new graphical interface to the migration tools. The migration tools are described in ″WASPreUpgrade command″ and ″WASPostUpgrade command″ in the information center.

Exit This option closes the First steps console.

Location of the command file

Installing the product creates a default profile for the server1 application server. The location of the First steps console for the default profile is:

- Linux UNIX *install_root*`/profiles/default/firststeps/firststeps.sh`
- Windows *install_root*`\profiles\default\firststeps\firststeps.bat`

The location of the `firststeps.sh` or `firststeps.bat` script for any profile is:

- Linux UNIX *install_root*`/profiles/`*profile_name*`/firststeps/firststeps.sh`
- Windows *install_root*`\profiles\`*profile_name*`\firststeps\firststeps.bat`

Parameters

No parameters are associated with this command.

Syntax for the firststeps command

Use the following syntax for the command:

- Linux UNIX `./firststeps.sh`

- Windows `firststeps.bat`

Usage tips

The following links exist on the First steps console for the WebSphere Application Server - Express product:

Option	Link
Installation verification	Calls the **ivt** command. The location of the installation verification test varies per platform: • Linux UNIX *install_root*/profiles/*profile_name*/bin/ivt.sh • Windows *install_root*\profiles\profile_name\bin\ivt.bat
Start the server	Calls the **startServer** command. The location of the **startServer** command varies per platform: • Linux server1 UNIX *install_root*/profiles/*profile_name*/bin/startServer.sh • Windows *install_root*\profiles\profile_name\bin\startServer.bat server1 When you have more than one application server on the same machine, the command starts the same application server that is associated with the First steps console.
Stop the server	Calls the **stopServer** command. The location of the **stopServer** command varies per platform: • Linux server1 UNIX *install_root*/profiles/*profile_name*/bin/stopServer.sh • Windows *install_root*\profiles\profile_name\bin\stopServer.bat server1
Administrative console	Opens the default browser to the http://localhost:9060/ibm/console Web address. When you have more than one application server on the same machine, the port varies. The First steps console starts the administrative console that is associated with the First steps console.

Option	Link
Profile creation wizard	Calls the **pct*platform*** command. The command is in the *install_root*/bin/ProfileCreator directory. The name of the command varies per platform: • AIX pctAIX.bin • HP-UX pctHPUX.bin • HP-UX 64-bit platforms: pctHPUXIA64.bin • Linux pctLinux.bin • Linux 64-bit platforms: pct.bin • Linux Power platforms: pctLinuxPPC.bin • Solaris pctSolaris.bin • Windows pctWindows.exe • Windows 64-bit platforms: pctWindowsIA64.exe
Samples Gallery	Opens the default browser to the http://localhost:9080/WSsamples Web address. If you do not install Samples, the option does not appear on the First steps console. If you do not install the Samples during the initial installation of the product, the option does not display on the First steps console. You can perform an incremental installation to add the Samples feature. After adding the Samples, the options displays on the First steps console. When you have more than one profile on the same machine, the port varies. The First steps console starts the Samples gallery that is associated with the First steps console.
Information center for WebSphere Application Server products	Opens the default browser to the online information center at the http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp Web address.
Migration wizard	Calls the **migration** command. The location of the **migration** command is: • Linux UNIX *install_root*/bin/migration.sh • Windows *install_root*\bin\migration.bat The migration tools are also in the /migration folder on the product disc.

Using the Profile creation wizard

This topic describes how to create run-time environments for WebSphere Application Server. Each run-time environment is created within a *profile*. A profile is the set of files that define the run-time environment. The Profile creation wizard creates the profile for each run-time environment.

Before using the Profile creation wizard, install the core product files.

The Profile creation wizard is the wizard interface to the profile creation tool, wasprofile. See the description of the "wasprofile command" on page 43 for more information.

An error can occur when you have not provided enough system temporary space to create a profile. Verify that you have a minimum of 40 MB of temp space available before creating a profile.

You must have 200 MB of available disk space in the directory where you create an Application Server profile.

AIX Manually verify that the required space for creating a profile is available on AIX. A known problem in the underlying InstallShield for Multiplatforms (ISMP) code prevents proper space checking on AIX systems at the time that the product disc was created.

Important: Concurrent profile creation is not supported at this time for one set of core product files. Concurrent attempts to create profiles result in a warning about a profile creation already in progress.

The installation procedure creates one profile named `default` for an application server named `server1`. You can use the Profile creation wizard to create more application server processes. For example, a second profile can allow two different teams in a department to test independently of one another using the same machine.

Each use of the Profile creation wizard or the **wasprofile** command line tool creates one profile.

1. Install the product to create the core product files.
2. Start the Profile creation wizard to create a new run-time environment.

   Several ways exist to start the wizard.

   One way to start the wizard is to issue the command directly from a command line.

   The command is in the *install_root*`/bin/ProfileCreator` directory. The name of the command varies per platform:

   - AIX `pctAIX.bin`
   - HP-UX `pctHPUX.bin`
   - HP-UX 64-bit platforms: `pctHPUXIA64.bin`
   - Linux `pctLinux.bin`
   - Linux 64-bit platforms: `pct.bin`
   - Linux Power platforms: `pctLinuxPPC.bin`
   - Solaris `pctSolaris.bin`
   - Windows `pctWindows.exe`
   - Windows 64-bit platforms: `pctWindowsIA64.exe`

   Another way to start the Profile creation wizard is to select the wizard from the First steps console.

   a. Open a command window.

   b. Change directories to the `firststeps` directory in the installation root directory:

      The installation root varies by platform:

      - AIX `/usr/IBM/WebSphere/AppServer/firststeps`
      - HP-UX Linux Solaris `/opt/IBM/WebSphere/AppServer/firststeps`
      - Windows `C:\Program Files\IBM\WebSphere\AppServer\firststeps`

   c. Issue the **firststeps** command to start the console:

        - Linux UNIX `./firststeps.sh`
        - Windows `firststeps.bat`
    d. Select the Profile creation wizard option on the console.

       The Profile creation wizard is an InstallShield for Multiplatforms application. The wizard loads the Java 2 SDK and then displays its Welcome panel.

    See the description of the “firststeps command” on page 33 for more information.
3. Create another stand-alone application server.

   See “Using the Profile creation wizard to create an application server.”

   The installation procedure creates a stand-alone application server during installation. However, you can use the Profile creation wizard to create additional stand-alone application servers.

See the description of the “wasprofile command” on page 43 to learn more about the command-line alternative method of creating a profile, and to see examples of using the command.

See “Planning the installation (diagrams)” on page 15 for examples of configurations that you can create by creating profiles.

Using the Profile creation wizard to create an application server

The Profile creation wizard can create an application server profile on any machine where the core product files exist.

Before using the Profile creation wizard, install the core product files.

The Profile creation wizard is the wizard interface to the profile creation tool, wasprofile. See the description of the “wasprofile command” on page 43 for more information.

An error can occur when you have not provided enough system temporary space to create a profile. Verify that you have a minimum of 40 MB of temp space available before creating a profile.

You must have 200 MB of available disk space in the directory where you create an Application Server profile.

AIX Manually verify that the required space for creating a profile is available on AIX. A known problem in the underlying InstallShield for Multiplatforms (ISMP) code prevents proper space checking on AIX systems at the time that the product disc was created.

The Installation wizard creates an application server profile with a server named server1. You can create additional profiles. Each additional profile is an application server named server1.

This procedure describes creating an application server profile using the graphical user interface provided by the Profile creation wizard.

You can also use the **wasprofile** command to create an application server profile. See the description of the “wasprofile command” on page 43 for more information.

1. Start the Profile creation wizard to create a new run-time environment.

   Several ways exist to start the wizard.

   One way to start the wizard is to issue the command directly from a command line.

   The command is in the *install_root*/`bin/ProfileCreator` directory. The name of the command varies per platform:

   - AIX `pctAIX.bin`

- HP-UX `pctHPUX.bin`
- HP-UX 64-bit platforms: `pctHPUXIA64.bin`
- Linux `pctLinux.bin`
- Linux 64-bit platforms: `pct.bin`
- Linux Power platforms: `pctLinuxPPC.bin`
- Solaris `pctSolaris.bin`
- Windows `pctWindows.exe`
- Windows 64-bit platforms: `pctWindowsIA64.exe`

Another way to start the Profile creation wizard is to select the wizard from the First steps console.

a. Open a command window.

b. Change directories to the `firststeps` directory in the installation root directory:

   The installation root varies by platform:

   - AIX `/usr/IBM/WebSphere/AppServer/firststeps`
   - HP-UX Linux Solaris `/opt/IBM/WebSphere/AppServer/firststeps`
   - Windows `C:\Program Files\IBM\WebSphere\AppServer\firststeps`

c. Issue the **firststeps** command to start the console:

   - Linux UNIX `./firststeps.sh`
   - Windows `firststeps.bat`

d. Select the Profile creation wizard option on the console.

   The Profile creation wizard is an InstallShield for Multiplatforms application. The wizard loads the Java 2 SDK and then displays its Welcome panel.

See the description of the "firststeps command" on page 33 for more information.

2. Click **Next** on the Welcome panel.

   The wizard displays the Profile type selection panel.

3. Click **Next**.

   The wizard displays the Profile name panel.

   Each profile that you create must have a name. The name is the name of the folder that contains all of the files that define the run-time environment for the profile. When you have more than one profile, you can tell them apart at their highest level by this name.

4. Specify a name for the profile, then click **Next**.

   Profile naming guidelines: The profile name can be any unique name with the following restrictions. Do not use any of the following characters when naming your profile:

   - Spaces
   - Illegal special characters that are not allowed within the name of a directory on your operating system, such as *&?
   - Slashes (/) or (\)

   Double-byte characters are allowed.

   The default profile

   The first profile that you create on a machine is the default profile. The default profile is the default target for commands issued from the `bin` directory in the product installation root. When only one profile exists on a machine, every command works on the only server process in the configuration.

Addressing a profile in a multi-profile environment

When two or more profiles exist on a machine, certain commands require that you specify the profile to which the command applies. These commands use the -profileName parameter to identify which profile to address. You might find it easier to use the commands that in the `bin` directory of each profile.

A command in the `profiles/`*`profile_name`*`/bin` directory has two lines. The first line sets the WAS_USER_SCRIPT environment variable for the command window. The variable sets up the command environment to address the profile. The second line calls the actual command in the *install_root*/`bin` directory.

The actual command queries the command shell to determine the calling profile and to autonomically address the command to the calling profile.

The wizard then displays the Profile directory panel.

5. Accept the default directory or specify a non-default location, then click **Next**. Or click **Browse** to select a different location.

   If you click **Back** and change the name of the profile, you must manually change the name on this panel when it displays again.

   The wizard displays the Node and host name panel.

6. Specify the characteristics for the application server, then click **Next**.

   Use unique names for each application server that you create.

   Reserved names: Avoid using reserved folder names as field values. The use of reserved folder names can cause unpredictable results. The following words are reserved:

   - cells
   - nodes
   - servers
   - clusters
   - applications
   - deployments

Field name	Default value	Constraints	Description
Node name	Name of your machine	Avoid using the reserved words.	Pick any name you want. To help organize your installation, use a unique name if you plan to create more than one application server on the machine.
Host name	DNS name of your machine	Addressable through your network.	Use the actual DNS name or IP address of your machine to enable communication with your machine. See additional information about the host name following this table.

Node name considerations:

Windows The installation directory path must be no longer than 60 characters.

Host name considerations:

The host name is the network name for the physical machine on which the node is installed. The host name must resolve to a physical network node on the server. When multiple network cards exist in the server, the host name or IP address must resolve to one of the network cards. Remote nodes use the host name to connect to and to communicate with this node. Selecting a host name that other machines can reach within your network is extremely important. Do not use the generic localhost identifier for this value.

If you define coexisting nodes on the same computer with unique IP addresses, define each IP address in a domain name server (DNS) look-up table. Configuration files for stand-alone Application Servers do not provide domain name resolution for multiple IP addresses on a machine with a single network address.

The value that you specify for the host name is used as the value of the hostName property in configuration documents for the stand-alone Application Server. Specify the host name value in one of the following formats:

- Fully qualified domain name servers (DNS) host name string, such as xmachine.manhattan.ibm.com
- The default short DNS host name string, such as xmachine
- Numeric IP address, such as 127.1.255.3

The fully qualified DNS host name has the advantage of being totally unambiguous and also flexible. You have the flexibility of changing the actual IP address for the host system without having to change the Application Server configuration. This value for host name is particularly useful if you plan to change the IP address frequently when using Dynamic Host Configuration Protocol (DHCP) to assign IP addresses. A format disadvantage is being dependent on DNS. If DNS is not available, then connectivity is compromised.

The short host name is also dynamically resolvable. A short name format has the added ability of being redefined in the local hosts file so that the system can run the Application Server even when disconnected from the network. Define the short name to 127.0.0.1 (local loopback) in the hosts file to run disconnected. A format disadvantage is being dependent on DNS for remote access. If DNS is not available, then connectivity is compromised.

A numeric IP address has the advantage of not requiring name resolution through DNS. A remote node can connect to the node you name with a numeric IP address without DNS being available. A format disadvantage is that the numeric IP address is fixed. You must change the setting of the hostName property in Express configuration documents whenever you change the machine IP address. Therefore, do not use a numeric IP address if you use DHCP, or if you change IP addresses regularly. Another format disadvantage is that you cannot use the node if the host is disconnected from the network.

After specifying application server characteristics, the wizard displays the Port value assignment panel.

7. Verify that the ports specified for the stand-alone application server are unique, then click **Next**.

   Windows After specifying port assignments, the wizard displays the Windows service definition panel, if you are installing on a Windows platform.

8. Windows Choose whether to run the application server as a Windows service on a Windows platform and click **Next**.

   Version 6 attempts to start Windows services for application server processes started by a **startServer** command. For example, if you configure an application server as a Windows service and issue the **startServer** command, the **wasservice** command attempts to start the defined service.

   If you chose to install a local system service, you do not have to specify your user ID or password. If you create a specified user type of service, you must specify the user ID and the password for the user who is to run the service. The user must have *Log on as a service* authority for the service to run properly.

   To perform this installation task, the user ID must not have spaces in its name. The ID must also belong to the administrator group and must have the advanced user rights *Act as part of the operating system* and *Log on as a service*. The Installation wizard grants the user ID the advanced user rights if it does not already have them, if the user ID belongs to the administrator group.

   You can also create other Windows services after the installation is complete, to start other server processes. See “Automatically restarting server processes” on page 149 for more information.

   The installation wizard shows which components are selected for installation in a pre-installation summary panel.

9. Click **Next** to create the application server or click **Back** to change the characteristics of the application server.

   The wizard displays the Installation status panel that shows which components are installing.

   When the installation is complete, the wizard displays the Profile creation is complete panel.
10. Click **Finish** to exit, then click **Profile creation wizard** on the First steps console to start the wizard again to create other application servers.

You can create an application server profile. The node within the profile has an application server named server1.

Refer to the description of the "wasprofile command" on page 43 to learn about creating this type of profile using a command instead of a wizard.

Deploy an application to get started!

See Fast paths for WebSphere Application Server to get started deploying applications.

Deleting a profile

This topic describes how to manually delete a profile.

Before using the manual procedure to remove a profile, try the **wasprofile** command with the -delete option. For example, issue one of the following commands:

Linux UNIX

```
./wasprofile.sh -delete
                -profileName profile_name | -profilePath profile_path
```

Windows

```
wasprofile.bat -delete
               -profileName profile_name | -profilePath profile_path
```

See "wasprofile command" on page 43.

If the command does not work, use this procedure to delete the profile.

This procedure describes how to manually delete a profile when the **wasprofile -delete** command results in the following message:

```
INSTCONFFAILED: Cannot delete profile
```

1. Delete the *profiles_install_root*/*profile_name* directory.
2. If the *install_root*/`properties/profileRegistry.xml` file exists, edit the file in a flat-file editor to delete the entry for the profile, if the entry is present.

   The entry resembles the following example:

   ```
   <profile isDefault="true"
            name="BadProfile"
            path="E:\IBM\WebSphere\AppServer\profiles\BadProfile"
            template="E:\IBM\WebSphere\AppServer\profileTemplates\default"/>
   ```
3. Linux UNIX Compare the two batch files, *install_root*/ `properties/ fsdb/ _was_profile_default/ default.sh` and *install_root*/ `properties/ fsdb/` *`bad_profile_name`*`.sh`.

   If the files are identical, delete the *install_root*/ `properties/ fsdb/ _was_profile_default` directory and the *install_root*/ `properties/ fsdb/` *`bad_profile_name`*`.sh` file.

   If the files are not identical, delete only the *install_root*/ `properties/ fsdb/` *`bad_profile_name`*`.sh` file.

4. Windows Compare the two batch files, *install_root*\ `properties\ fsdb\ _was_profile_default\ default.bat` and *install_root*\ `properties\ fsdb\` *`bad_profile_name`*`.bat`.

   If the files are identical, delete the *install_root*\ `properties\ fsdb\ _was_profile_default` directory and the *install_root*\ `properties\ fsdb\` *`bad_profile_name`*`.bat` file.

   If the files are not identical, delete only the *install_root*\ `properties\ fsdb\` *`bad_profile_name`*`.bat` file.

See the description of the “wasprofile command” to learn more about the command-line method of working with profiles.

See “Using the Profile creation wizard” on page 36 for more information about creating profiles with the Profile creation wizard.

wasprofile command

The **wasprofile** command line tool creates all Application Server run-time environments in Version 6. The command creates a profile, which is the set of files that define the run-time environment for a stand-alone Application Server.

The **wasprofile** command is also referred to as the *profile creation tool*.

Introduction to terms that describe Version 6 profiles

The **wasprofile** command creates the run-time environment for a WebSphere Application Server process in a set of files called a *profile*. The profile defines the run-time environment and includes all of the files that the server processes in the run-time environment can change. The *profile creation tool* and its graphical user interface, the *Profile creation wizard*, are the only ways to create run-time environments in V6.

The *Profile creation wizard* is an InstallShield for Multiplatforms (ISMP) application. You can use the wizard to enter most of the parameters that are described in this topic. Some parameters, however, require you to use the **wasprofile** command. You must use the **wasprofile** command to delete a profile, for instance, because the Profile creation wizard does not provide a deletion function.

However, the Profile creation wizard also performs tasks that the wasprofile command does not. For instance, the wizard can create a Windows service for each profile that it creates. It can also assign non-conflicting ports based on previous Version 6 port assignments.

Core product files: The core product files are the shared product binaries. The binary files are shared by all profiles.

The directory structure for V6 has two major divisions of files in the installation root directory for the product:

- The core product files are shared product binary files that do not change unless you install a refresh pack, a fix pack, or an interim fix. Some log information is also updated.
- The `profiles` directory is the default directory for creating profiles. The configuration for every defined Application Server process is within the `profiles` directory unless you specify a new directory when you create a profile. These files change as often as you create a new profile, reconfigure an existing profile, or delete a profile.

All of the folders except for the `profiles` directory and a few others such as the `logs` directory and the `properties` directory do not change unless you install service fixes. The `profiles` directory, however, changes each time you add, change, or delete a profile. The profiles directory is the default repository for profiles. However, you can put a profile anywhere on the machine provided there is enough available disk space.

If you put a profile in another existing folder in the installation root directory, a risk exists that the profile might be affected by the installation of a service fix that applies maintenance to the folder. Use a directory outside of the installation root directory when using a directory other than the `profiles` directory for creating profiles.

WebSphere Application Server profile: The **wasprofile** command line tool defines each Application Server instance of a Version 6 product.

You must run the wizard or the command line tool each time that you want to create a stand-alone Application Server. A need for more than one stand-alone Application Server on a machine is common.

Administration is greatly enhanced when using V6 profiles instead of multiple product installs. Not only is disk space saved, but updating the product is simplified when you only maintain a single set of product core files. Also, creating new profiles is faster and less prone to error than full product installs, allowing a developer to create new disposable profiles of the product for development and testing.

You can run the Profile creation wizard or the profile creation tool to create a new Application Server environment on the same machine as an existing one. Simply define unique characteristics (such as profile name and node name) for the new profile. Each profile has its own administrative console and administrative scripting interface. Each Application Server process shares all run-time scripts, libraries, the Software Development Kit, and other core product files.

The installation program for WebSphere Application Server - Express uses the profile creation tool to create an Application Server profile named `default`.

Installed file set: You decide where to install the files that define a profile. The default location is in the `profiles` directory in the installation root directory. But you can change the location on the Profile creation wizard or in a parameter when using the command line tool. For example, assume that you create two profiles on a Linux platform with host name devhost1. The profile directories resemble the following example if you do not relocate them:

```
/opt/IBM/WebSphere/AppServer/profiles/devhost1Profile01
/opt/IBM/WebSphere/AppServer/profiles/devhost1Profile02
```

Suppose that you specify a different directory, such as `/opt/profiles`, for the profile directory field in the wizard. The profile directories resemble the following example:

```
/opt/profiles/devhost1Profile01
/opt/profiles/devhost1Profile02
```

The following directories exist within a profile. This example assumes that a profile named devhost1Profile01 exists:

/opt/IBM/WebSphere/AppServer/profiles/devhost1Profile01/bin

/opt/IBM/WebSphere/AppServer/profiles/devhost1Profile01/config

/opt/IBM/WebSphere/AppServer/profiles/devhost1Profile01/etc

/opt/IBM/WebSphere/AppServer/profiles/devhost1Profile01/firststeps

/opt/IBM/WebSphere/AppServer/profiles/devhost1Profile01/installableApps

/opt/IBM/WebSphere/AppServer/profiles/devhost1Profile01/installedApps

/opt/IBM/WebSphere/AppServer/profiles/devhost1Profile01/installedConnectors

/opt/IBM/WebSphere/AppServer/profiles/devhost1Profile01/installedFilters

/opt/IBM/WebSphere/AppServer/profiles/devhost1Profile01/logs

/opt/IBM/WebSphere/AppServer/profiles/devhost1Profile01/properties

/opt/IBM/WebSphere/AppServer/profiles/devhost1Profile01/samples

/opt/IBM/WebSphere/AppServer/profiles/devhost1Profile01/temp

/opt/IBM/WebSphere/AppServer/profiles/devhost1Profile01/tranlog

```
/opt/IBM/WebSphere/AppServer/profiles/devhost1Profile01/wstemp
```

The profile repository: The profile repository is the default location of profile-related metadata. The repository is the default location for new profiles, which is often referred to as the profiles installation root directory.

However, you can decide where to install a profile. The default location of the profile repository is the *install_root*/`profiles` directory. In the earlier example, creating two profiles on a Linux platform with host name devhost1 results in the following example directories in the profile repository:

```
/opt/IBM/WebSphere/AppServer/profiles/devhost1Profile01
/opt/IBM/WebSphere/AppServer/profiles/devhost1Profile02
```

When you specify a directory, such as `/opt/profiles`, the profiles are no longer in the default repository, which is not a problem. For example, the following locations are valid:

```
/opt/profiles/devhost1Profile01
/opt/profiles/devhost1Profile02
```

Location of the command file

The command file is located in the `*install_root*/bin` directory. The command file is a script named `wasprofile.sh` for Linux and UNIX platforms or `wasprofile.bat` for Windows platforms.

The Profile creation wizard is the graphical user interface to the command line tool. The file name of the command that calls the Profile creation wizard varies per operating system platform. See “Using the Profile creation wizard” on page 36 for more information.

Logging

The **wasprofile** command creates a log for every profile that it creates. The logs are in the *install_root*/`logs/wasprofile` directory. The files are named in this pattern: `wasprofile_create_`*`profile_name`*`.log`.

The command also creates a log for every profile that it deletes. The logs are in the *install_root*/`logs/wasprofile` directory. The files are named in this pattern: `wasprofile_delete_`*`profile_name`*`.log`.

Required disk space

AIX Manually verify that the required space for creating a profile is available on AIX. A known problem in the underlying InstallShield for Multiplatforms (ISMP) code prevents proper space checking on AIX systems at the time that the product disc was created.

An error can occur when you have not provided enough system temporary space to create a profile. Verify that you have a minimum of 40 MB of temp space available before creating a profile.

You must have 200 MB of available disk space in the directory where you create an Application Server profile.

Concurrent profile creation

Important: Concurrent profile creation is not supported at this time for one set of core product files. Concurrent attempts to create profiles result in a warning about a profile creation already in progress.

Entering lengthy commands on more than one line

The length of the **wasprofile** command can exceed the normal shell window limit for one line of 256 characters. If your command is longer than the limit, issue the command on multiple lines by ending a line with a backward slash, pressing **Enter**, and continuing the command on the next line.

For example, on a Solaris system, the following command requires input on multiple lines:

```
./wasprofile.sh \
-create -profileName bladetcb6profile \
-profilePath /usr/IBM/WebSphere/AppServer/profiles/bladetcb6profile \
-templatePath /usr/WebSphere/AppServer/profileTemplates/default \
-nodeName bladetcb6node \
-cellName bladetcb6Cell \
-hostName bladetcb6.rtp.raleigh.ibm.com
```

Omit the line continuation character from the last line to signal the end of the command to the operating system.

wasprofile.sh command syntax

List existing profiles:

```
# ./wasprofile.sh -listProfiles
                 [-debug]
```

Delete profiles:

```
# ./wasprofile.sh -delete
                -profileName profile_name | -profilePath profile_path
               [-debug]
```

Create new profiles:

```
wasprofile.sh -create
              -profileName profile_name
              -profilePath fully_qualified_profile_path
              -templatePath template_path
              -nodeName node_name
              -cellName cell_name
              -hostName host_name
              -server  iSeries_server_name
             [-startingPort starting_port | -portsFile filepath]
              -winserviceCheck true | false
              -winserviceAccountType specifieduser | localsystem
              -winserviceUserName yourusername
              -winservicePassword yourpassword
              -winserviceStartupType manual | automatic | disabled
             [-debug]
```

Get name of existing profile from path:

```
# ./wasprofile.sh -getName
                 -profilePath profile_path
                [-debug]
```

Get path of existing profile from name:

```
# ./wasprofile.sh -getPath
                 -profileName profile_name
                [-debug]
```

Check the integrity of the profile registry:

```
# ./wasprofile.sh -validateRegistry
                [-debug]
```

Check the integrity of the profile registry, removing profiles that are not found:

```
# ./wasprofile.sh -validateAndUpdateRegistry
                 [-backup file_name]
                 [-debug]
```

wasprofile.bat command syntax

List existing profiles:

```
wasprofile.bat -listProfiles
             [-debug]
```

Delete profiles:

```
wasprofile.bat -delete
              -profileName profile_name | -profilePath profile_path
             [-debug]
```

Create new profiles:

```
wasprofile.bat -create
              -profileName profile_name
              -profilePath fully_qualified_profile_path
              -templatePath template_path
              -nodeName node_name
             [-cellName cell_name]
              -hostName host_name
              -server  iSeries_server_name
             [-startingPort starting_port | -portsFile filepath]
              -winserviceCheck true | false
              -winserviceAccountType specifieduser | localsystem
              -winserviceUserName yourusername
              -winservicePassword yourpassword
              -winserviceStartupType manual | automatic | disabled
             [-debug]
```

When the -startingPort parameter is not used, the profile creation tool uses the default port settings specified in the `serverindex.xml` file.

Get name of existing profile from path:

```
wasprofile.bat -getName
              -profilePath fully_qualified_profile_path
             [-debug]
```

Get path of existing profile from name:

```
wasprofile.bat -getPath
              -profileName profile_name
             [-debug]
```

Check integrity of profile registry:

```
wasprofile.bat -validateRegistry
             [-debug]
```

Check integrity of profile registry, removing unfound profiles:

```
wasprofile.bat -validateAndUpdateRegistry
             [-backup file_name]
             [-debug]
```

Parameters

Supported arguments include:

-augment
Refreshes or augments the given profile using the template in the templatePath parameter.

-backup *file_name*
Backs up the profile registry file to a file with the file name specified.

-cellname *file_name*
Specifies the cell name of the profile.

This is an optional parameter. If you omit the parameter, a default cell name is assigned.

-create
Creates the profile.

-debug
Turns on the debug function of the Ant utility, which the **wasprofile** command uses.

-delete
Deletes the profile.

-getName
Gets the name for a profile registered at a given file system path. Requires the –profilePath parameter.

-getPath
Gets the file system location for a profile of a given name. Requires the –profileName parameter.

-hostName *host_name*
Specifies the host name where you are creating the profile. This should match the host name that you specified during installation of the initial product.

-listProfiles
Llists all defined profiles.

-nodeName *node_name*
Specifies the node name for the node that is created with the new profile. Use a unique value or on the machine. Each profile that shares the same set of product binaries must have a unique node name.

-portsFile *file_path*
An optional parameter that specifies the path to a file that defines port settings for the new profile. When omitted, the wasprofile tool looks for the *install_root* `/profileTemplates/`*`profile_type`* `/actions/portsUpdate/bin/portdef.props` file.

Do not use this parameter when using the startingPort parameter.

-profileName *profile_name*
Specifies the name of the profile. Use a unique value when creating a profile. Each profile that shares the same set of product binaries must have a unique name.

-profilePath *profile_path*
Specifies the fully qualified path to the profile.

Windows If the fully qualified path contains spaces, enclose the value in quotation marks.

Specifies the name of the server on an iSeries platform.

-startingPort *startingPort*
Specifies the starting port number for generating all ports for the profile. If not specified, the **wasprofile** command uses default ports specified in the `serverindex.xml` file.

-templatePath *template_path*
Specifies the path to the templates in the shared binaries.

-validateAndUpdateRegistry *registry_file backup_file*
Checks all of the profiles that are listed in the profile registry to see if the profiles are present on the file system. Removes any missing profiles from the registry. Returns a list of the missing profiles that were deleted from the profile.

-validateRegistry *registry_file*
Checks all of the profiles that are listed in the profile registry to see if the profiles are present on the file system. Returns a list of missing profiles.

Windows **-winserviceAccountType** *type_of_owner_account*

The type of the owner account of the Windows service created for the profile can be either specifieduser or localsystem. The Windows service can run under the local account of the user who is creating the profile.

Windows **winserviceCheck** *value*

The value can be either true or false. Specify `true` to create a Windows service for the server process that is created within the profile. Specify `false` to not create the Windows service.

Windows **-winservicePassword** *yourpassword*

Specify the password for the specified user or the local account that is to own the Windows service.

Windows **-winserviceStartupType** *startup_type*

Possible startup_type values are:

- manual
- automatic
- disabled

See ″WASService command″ in the Installation PDF for more information about Windows services.

Windows **-winserviceUserName** *user_ID*

Specify your user ID so that Windows can verify you as an ID that is capable of creating a Windows service. Your user ID must belong to the administrator group and have the following advanced user rights, *Act as part of the operating system* and *Log on as a service*

Use case scenarios

Use cases are a description of common tasks for which the tool is used.

Scenario: Deleting a profile: The following command is on more than one line for clarity. Enter the command on one line to delete the profile named `shasti`:

```
wasprofile.sh -delete
              -profileName shasti
```

Scenario: Using predefined port numbers: When you use the wasprofile tool without the -startingPort parameter, the tool uses the `/profileTemplates/`*`profile_type`* `/actions/portsUpdate/bin/portdef.props` file to set the initial ports.

Example of using the -portsFile parameter

Copy the file, edit the port settings, and use your copy by using the -portsFile parameter as shown in the following example:

```
wasprofile.bat
   -create
   -profileName Wow_Profile
   -profilePath
       C:\ExpressV6\IBM\WebSphere\AppServer\profiles\Wow_Profile
   -templatePath
       C:\ExpressV6\IBM\WebSphere\AppServer\profileTemplates\default
   -nodeName Wow_node
   -cellName Wow_cell
   -hostName loyAllen
   -portsFile C:\temp\ports\portdef.props
```

Suppose that the `portdef.props` file has the following values:

```
WC_defaulthost=39080
WC_adminhost=39060
WC_defaulthost_secure=39443
```

```
WC_adminhost_secure=39043
BOOTSTRAP_ADDRESS=32809
SOAP_CONNECTOR_ADDRESS=38880
SAS_SSL_SERVERAUTH_LISTENER_ADDRESS=39401
CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS=39403
CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS=39402
ORB_LISTENER_ADDRESS=39100
DCS_UNICAST_ADDRESS=39353
SIB_ENDPOINT_ADDRESS=37276
SIB_ENDPOINT_SECURE_ADDRESS=37286
SIB_MQ_ENDPOINT_ADDRESS=35558
SIB_MQ_ENDPOINT_SECURE_ADDRESS=35578
```

As you run the command, messages similar to the following appear in the output stream:

```
replaceRegExpAllInstancesOfGivenTokenWithGivenValueForTheGivenFile:
  [echo] File C:\ExpressV6\IBM\WebSphere\AppServer\profiles\
  Wow_Profile/config/templates/default/serverentry-template.xml:
  setting CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS to 39403

...
replaceRegExpAllInstancesOfGivenTokenWithGivenValueForTheGivenFile:
   [echo] File C:\ExpressV6\IBM\WebSphere\AppServer\profiles\
   Wow_Profile/config/templates/default/serverentry-template.xml:
   setting CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS to 39402
...
```

The resulting serverindex.xml file looks similar to the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<serverindex:ServerIndex xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
...
   <specialEndpoints xmi:id="NamedEndPoint_..."
                                         endPointName="BOOTSTRAP_ADDRESS">
      <endPoint xmi:id="EndPoint_..." host="IBMmachine" port="32809"/>
    </specialEndpoints>
    <specialEndpoints xmi:id="NamedEndPoint_..."
                                   endPointName="SOAP_CONNECTOR_ADDRESS">
      <endPoint xmi:id="EndPoint_..." host="IBMmachine" port="38880"/>
    </specialEndpoints>
    <specialEndpoints xmi:id="NamedEndPoint_..."
                   endPointName="SAS_SSL_SERVERAUTH_LISTENER_ADDRESS">
      <endPoint xmi:id="EndPoint_..." host="IBMmachine" port="39401"/>
    </specialEndpoints>
    <specialEndpoints xmi:id="NamedEndPoint_..."
                  endPointName="CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS">
      <endPoint xmi:id="EndPoint_..." host="IBMmachine" port="39403"/>
    </specialEndpoints>
    <specialEndpoints xmi:id="NamedEndPoint_..."
                 endPointName="CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS">
      <endPoint xmi:id="EndPoint_..." host="IBMmachine" port="39402"/>
    </specialEndpoints>
    <specialEndpoints xmi:id="NamedEndPoint_..."
                                      endPointName="WC_adminhost">
      <endPoint xmi:id="EndPoint_..." host="*" port="39060"/>
    </specialEndpoints>
    <specialEndpoints xmi:id="NamedEndPoint_..."
                                    endPointName="WC_defaulthost">
      <endPoint xmi:id="EndPoint_..." host="*" port="39080"/>
    </specialEndpoints>
    <specialEndpoints xmi:id="NamedEndPoint_..."
                                  endPointName="DCS_UNICAST_ADDRESS">
      <endPoint xmi:id="EndPoint_..." host="IBMmachine" port="39353"/>
    </specialEndpoints>
    <specialEndpoints xmi:id="NamedEndPoint_..."
                                endPointName="WC_adminhost_secure">
      <endPoint xmi:id="EndPoint_..." host="*" port="39043"/>
    </specialEndpoints>
    <specialEndpoints xmi:id="NamedEndPoint_..."
                                endPointName="WC_defaulthost_secure">
      <endPoint xmi:id="EndPoint_..." host="*" port="39443"/>
    </specialEndpoints>
    <specialEndpoints xmi:id="NamedEndPoint_..."
                               endPointName="SIB_ENDPOINT_ADDRESS">
      <endPoint xmi:id="EndPoint_..." host="*" port="37276"/>
    </specialEndpoints>
    <specialEndpoints xmi:id="NamedEndPoint_..."
                         endPointName="SIB_ENDPOINT_SECURE_ADDRESS">
      <endPoint xmi:id="EndPoint_..." host="*" port="37286"/>
    </specialEndpoints>
    <specialEndpoints xmi:id="NamedEndPoint_..."
                              endPointName="SIB_MQ_ENDPOINT_ADDRESS">
      <endPoint xmi:id="EndPoint_..." host="*" port="35558"/>
    </specialEndpoints>
    <specialEndpoints xmi:id="NamedEndPoint_..."
                       endPointName="SIB_MQ_ENDPOINT_SECURE_ADDRESS">
      <endPoint xmi:id="EndPoint_..." host="*" port="35578"/>
    </specialEndpoints>
    <specialEndpoints xmi:id="NamedEndPoint_..."
                                endPointName="ORB_LISTENER_ADDRESS">
      <endPoint xmi:id="EndPoint_..." host="IBMmachine" port="39100"/>
    </specialEndpoints>
  </serverEntries>
</serverindex:ServerIndex>
```

The **wasprofile** command creates a copy of the current `portdefs.props` file in the *install_root*`\profiles\`*`profile_name`*`\logs` directory.

Do not use the portsFile parameter when using the startingPort parameter. The two parameters are mutually exclusive.

Scenario: Incrementing default port numbers from a starting point: The **wasprofile** command can assign port numbers based on a starting port value that you give on the command line, using the `-startingPort` parameter. The tool assigns port numbers sequentially from the starting port number value.

The order of port assignments is arbitrary. Predicting assignments is not possible.

For example, ports created with -startingPort 20002 would appear similar to the following example:

Assigned ports for an Application Server profile

```
WC_defaulthost=20002
WC_adminhost=20003
WC_defaulthost_secure=20004
WC_adminhost_secure=20005
BOOTSTRAP_ADDRESS=20006
SOAP_CONNECTOR_ADDRESS=20007
SAS_SSL_SERVERAUTH_LISTENER_ADDRESS=20008
CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS=20009
CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS=20010
ORB_LISTENER_ADDRESS=20011
DCS_UNICAST_ADDRESS=20012
SIB_ENDPOINT_ADDRESS=20013
SIB_ENDPOINT_SECURE_ADDRESS=20014
SIB_MQ_ENDPOINT_ADDRESS=20015
SIB_MQ_ENDPOINT_SECURE_ADDRESS=20016
```

Example of startingPort parameter use

The following example of using the **wasprofile** command creates ports from an initial value of 20002, with the content shown in the previous example:

```
wasprofile.bat -create
               -profileName shasti
               -profilePath G:\shasti\WebSphere
               -templatePath template_path
               -nodeName W2K03
               -cellName W2K03_Cell01
               -hostName planetnt
               -startingPort 20002
```

Scenario: Setting up and using the profile environment: Most tasks that you perform in a profile are done using the -profileName attribute on the command line tools that you use. Such a scenario might be:

1. Create the server process using the *install_root*/bin/wasprofile.sh (or wasprofile.bat) script from the original installation. Assume that you create the Profile02 profile.
2. In that command window or in another, change directories to the /bin directory of the new server process.
3. Establish a temporary override for the normal WebSphere Application Server environment by using the -profileName attribute on a command you issue. In the same window, start server1 by changing directories to the install_root/bin directory of the original installation and issuing the command. There is no such command in the /bin directory of the server process:

   ```
   startServer.sh server1 -profileName Profile02
   ```
4. Notice the changes in the environment. Display all of the ports for the machine to see the ports that you set for the server process. For example, in a Linux bash shell or in the command window on a Windows platform, type:

   ```
   netstat -a
   ```
5. Open a browser window and point it at the port defined for the HTTP_TRANSPORT_ADMIN port of the new process. For example, suppose the setting is HTTP_TRANSPORT_ADMIN=20003. Open the administrative console for server1 by pointing your browser at:

   ```
   http://hostname_orIP_address:20003/ibm/console/
   ```

Scenario: Profile creation for a non-root user: Two methods exist for a non-root user to create a profile:

- The root user creates the profile and assigns ownership to the non-root user.
- A non-root user creates a profile after getting write permission to the appropriate directories.

Remember: An ease-of-use limitation exists for non-root users who create profiles. Mechanisms within the Profile creation wizard that suggest unique names and port values are disabled for non-root users. The non-root user must change the default field values in the Profile creation wizard for the profile name, node name, and port assignments. Consider assigning non-root users a range of values for each of the fields. You can assign responsibility to the non-root profilers for adhering to their proper value ranges and for maintaining the integrity of their own definitions.

Root creates the profile and assigns ownership to a non-root user: The root user can create a profile and assigns ownership of the profile directory to a non-root user.

1. The root user creates the profile with the following command:

   ```
   ./wasprofile.sh -create -profileName profile01 -profilePath install_root/profiles/profile01
   ```
2. The root user changes ownership of the directory for the profile to the non-root user with the following command:

   ```
   chown -R user1 install_root/profiles/profile01
   ```

A non-root user creates the profile (advanced option): The root user can grant write permission to the appropriate files and directories to a non-root user. The non-root user can then create the profile. You can create a group for users who are authorized to create profiles. Or you can give everyone the ability to create profiles. The following example shows how to create a group that is authorized to create profiles.

1. Log on to the Application Server system as root.
2. Create the `profilers` group that you can use to create profiles.
3. Create a user named `user1` to create profiles.
4. Add users root and `user1` to the `profilers` group.
5. Log off and back on as root to pick up the new group.
6. As root, use operating system tools to change file permissions.

   The following example assumes that the installation root directory is `/opt/IBM/WebSphere/AppServer`:

   ```
   mkdir /opt/IBM/WebSphere/AppServer/logs/wasprofile
   chgrp profilers /opt/IBM/WebSphere/AppServer/logs/wasprofile
   chmod g+wr  /opt/IBM/WebSphere/AppServer/logs/wasprofile
   chgrp profilers /opt/IBM/WebSphere/AppServer/properties
   chmod g+wr  /opt/IBM/WebSphere/AppServer/properties
   chmod g+wr  /opt/IBM/WebSphere/AppServer/properties/profileRegistry.xml
   ```

   You might have to change the permissions on additional `/opt/IBM/WebSphere/AppServer` directories if you encounter permission problems.
7. The non-root user who belongs to the profilers group can then create a profile in any directory to which the non-root user has write permission.

   If the non-root user does not have write access to any directories, it is up to the root user to change that situation. If the non-root user does not have write access to the /tmp directory, it is up to the root user to change that as well.

   The commands listed in step 6 give users assigned to the profilers group the ability to write to the `/opt/IBM/WebSphere/AppServer/logs/wasprofile` directory and to the `/opt/IBM/WebSphere/AppServer/properties` directory. It is not necessary to write to any other directories in the installation root of your WebSphere Application Server product.

   Have non-root users create a `profiles` directory in their own area, not in the installation root directory of the product.

Using the installation verification test

This topic describes how to use the installation verification test (IVT). The IVT verifies that the installation of the application server profile was successful. A *profile* consists of files that define the run-time environment for an application server. Each profile has its own IVT command.

After installing the product, you are ready to use the installation verification test (IVT).

The IVT program scans product log files for errors and verifies core functionality of the product installation.

After installing the product, the Installation wizard displays a prompt for starting the First steps console.

Installation verification is the first option on the First steps console.

The test consists of starting and monitoring the application server during its start up.

1. Select **Installation verification** on the First steps console after installing the product.

   You can also start the First steps console from the command line, as described in “firststeps command” on page 33.

   You can also start the “ivt command” on page 54 directly from the `bin` directory of the profile:

   - Linux UNIX *install_root*`/profiles/default/bin/ivt.sh`
   - Windows *install_root*`\profiles\default\bin\ivt.bat`

   If you create additional profiles, the `ivt` script location is within the *profile_home*`/bin` directory.
2. Observe the results in the First steps status window.

   The log file for installation verification is the *install_root*`/profiles/default/logs/ivtClient.log` file. If you create additional profiles, the file path is *profile_home*`/logs/ivtClient.log`.

The IVT provides the following useful information about the application server:

- The application server name
- The name of the profile
- The profile file path
- The type of profile
- The node name
- The current encoding
- The port number for the administrative console
- Various informational messages that include the location of the `SystemOut.log` file and how many errors are listed within the file
- A completion message

As the IVT starts the application server on a Windows platform, the IVT attempts to start the Windows service for the application server, if a Windows service exists. This is true even though the Windows service might have a manual startup type.

See “Automatically restarting server processes” on page 149 for more information.

3. If the log shows that errors occurred during the installation verification, correct the errors and run the IVT again. If necessary, create a new profile after correcting the error, and run the IVT on the new profile.

The IVT program starts the server process automatically if the server is not running. Once the server initializes, the IVT runs a series of verification tests. The tool displays pass or fail status in a console window. The tool also logs results to the *profile_home*`/logs/ivtClient.log` file. As the IVT verifies your system, the tool reports any detectable errors in the `SystemOut.log` file.

Return to the ″Task overview: Installing″ topic in the information center to continue.

ivt command

The **ivt** command starts the installation verification test (IVT) program. The IVT verifies that the installation of the application server profile was successful. A *profile* consists of files that define the run-time environment for an application server. Each profile has its own IVT command.

The IVT program starts the application server automatically if the server process is not already running. After the server process initializes, the IVT runs a series of verification tests and displays pass or fail status in a console window.

The IVT program scans the `SystemOut.log` file for errors and verifies core functionality of the profile.

You can start the IVT program from the command line or from the First steps console.

Location of the command file

Installing the product creates a default profile for the server1 application server. The location of the installation verification test program for the default profile is:

- Linux UNIX *install_root*`/profiles/default/bin/ivt.sh`
- Windows *install_root*`\profiles\default\bin\ivt.bat`

The location of the `ivt.sh` or `ivt.bat` script for any profile is:

- Linux UNIX *install_root*`/profiles/`*profile_name*`/bin/ivt.sh`
- Windows *install_root*`\profiles\`*profile_name*`\bin\ivt.bat`

Parameters

The following parameters are associated with this command.

server_name
Required parameter that identifies the name of the server process, such as server1.

profile_name
Required parameter that identifies the name of the profile that contains the server definition.

-p *server_port_number*
Optional parameter that identifies the default_host port when the port is not 9080, which is the default.

-host *machine_host_name*
Optional parameter that identifies the host machine of the profile to test. The default is localhost.

Syntax for the ivt command

Use the following syntax for the command:

- Linux UNIX *install_root*/profiles/*profile_name*/bin/ivt.sh
- Windows *install_root*\profiles\profile_name\bin\ivt.bat

Logging

The **ivt** command logs results to the *install_root*/profiles/*profile name*/logs/ivtClient.log file.

Example

The following examples test the server1 process in the profile01 profile on the myhost machine using the default_host on port 9081.

Windows

```
ivt.bat server1 profile01 -p 9081 -host myhost
```

Linux UNIX

```
ivt.sh server1 profile01 -p 9081 -host myhost
```

Configuring ports

This topic discusses configuring ports, particularly in coexistence scenarios.

1. Review “Port number settings in WebSphere Application Server versions” on page 56.

   This topic provides reference information about identifying port numbers in versions of WebSphere Application Server, as a means of determining port conflicts that might occur when you intend for an earlier version to coexist with Version 6.
2. You can change port settings on the port assignment panel while using the Installation wizard orthe Profile creation wizard.

   See “Using the Profile creation wizard” on page 36 and the *Installing your application serving environment* PDF for more information.
3. After installation, edit the *profiles_install_root*/*profile_name*/config/cells/*cell_name*/nodes/*node_name*/serverindex.xml file to change the port settings, or use scripting to change the values.

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

Port number settings in WebSphere Application Server versions

This topic provides reference information about identifying port numbers in versions of WebSphere Application Server, as a means of determining port conflicts that might occur when you intend for an earlier version to coexist or interoperate with Version 6.

Version 6 port numbers

Table 3. Port definitions for WebSphere Application Server Version 6

Port name	WebSphere Application Server	File
	Value	
HTTP_TRANSPORT	9080	serverindex.xml and virtualhosts.xml
HTTP Admin Console Port (HTTP_TRANSPORT_ADMIN)	9060	
HTTPS Transport Port (HTTPS_TRANSPORT)	9443	
HTTPS Admin Console Secure Port (HTTPS_TRANSPORT_ADMIN)	9043	
BOOTSTRAP_ADDRESS	2809	serverindex.xml
SOAP_CONNECTOR_ADDRESS	8880	
SAS_SSL_SERVERAUTH_LISTENER_ADDRESS	9401	
CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS	9403	
CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS	9402	
ORB_LISTENER_ADDRESS	9100	
DCS_UNICAST_ADDRESS	9353	
SIB_ENDPOINT_ADDRESS	7276	
SIB_ENDPOINT_SECURE_ADDRESS	7286	
SIB_MQ_ENDPOINT_ADDRESS	5558	
SIB_MQ_ENDPOINT_SECURE_ADDRESS	5578	
Internal JMS Server (JMSSERVER_SECURITY_PORT)	5557	
IBM HTTP Server Port	80	virtualhosts.xml,
IBM HTTP Server Admin Port	8008	*IHSinstall_root*/`conf/ admin.conf`
NODE_MULTICAST_IPV6_DISCOVERY_ADDRESS	5001	serverindex.xml

Version 5.x port numbers

Version 4.0.x port numbers

For WebSphere Application Server Advanced Single Server Edition, Version 4.0.x: Inspect the `server-cfg.xml` file to find the Web container HTTP transports port values for the configuration.

For WebSphere Application Server Advanced Edition, Version 4.0.x: When the administrative server is running, use this command to extract the configuration from the database:

```
xmlConfig -export config.xml -nodeName theNodeName
```

Look for the Web container HTTP transports port assignments.

Table 4. Port definitions for WebSphere Application Server V4.0.x

Port name	Value	Advanced Edition	IBM WebSphere Business Integration Server Foundation Edition	Advanced Single Server Edition
			File	
bootstrapPort	900	admin.config	admin.config	server-cfg.xml
lsdPort	9000			
LSDSSLPort	9001			
HTTP transport port	9080	database	database	
HTTPS transport port	9443			
Admin Console HTTP transport port	9090			
ObjectLevelTrace	2102			
diagThreadPort	7000			

Communicating with Web servers

The WebSphere Application Server works with a Web server to route requests for dynamic content, such as servlets, from Web applications. The Web servers are necessary for directing traffic from browsers to the applications that run in WebSphere Application Server. The Web server plug-in uses the XML configuration file to determine whether a request is from the Web server or the Application Server.

The Web server plug-ins for distributed platform Web servers are provided on a separate CD from the WebSphere Application Server products. A Web Server Plug-in Installation Wizard is also provided on that CD. Installing Web server plug-ins describes how to install a Web server plug-in and create a Web server definition.

1. Install your Web server if it is not already installed. If you want to use an IBM HTTP Server, see Installing IBM HTTP Server. Otherwise, see the installation information provided with your Web server.
2. Ensure that your Web server is configured to perform operations required by Web applications, such as GET and POST. Typically, this involves setting a directive in the Web server configuration file (such as the httpd.conf file for an IBM HTTP Server). Refer to the Web server documentation for instructions. If an operation is not enabled when a servlet or JSP file requiring the operation is accessed, an error message displays, such as this one from the IBM HTTP Server:

   ```
   HTTP method POST is not supported by this URL.
   ```
3. Use the Plug-in Installation wizard to install the appropriate plug-in file to your Web server and run the script configureWeb_server_name created by the wizard to create and configure the Web server definition in the WebSphere configuration repository. The following substeps are performed during the plug-in installation process. See the Plug-in Installation Roadmap for additional information.
   a. A Web server definitions is created. You can also use either use the administrative console or use the ConfigureWebServerDefintion.jacl script to create a Web server definition.

      If you use the administrative console:

      1) Select the node that was created in the preceding step, and in the Server name field, enter the local name of the Web server for which you are creating a Web server definition.
      2) Use the wizard to complete the Web server definition.

   b. An application or modules are mapped to a Web server. If an application that you want to use with this Web server is already installed, the application is automatically mapped to the Web server. If the application is not installed, select this Web server during the Map modules to servers step of the application installation process.

   c. Master repository is updated and saved.

4. **Optional:** Configure the plug-in. Use either the administrative console, or issue the “GenPluginCfg command” on page 688 to create your `plugin-cfg.xml` file.

   When setting up your Web server plug-in, you must decide whether or not to have the configuration automatically generated in response to a configuration change. When the Web server plug-in configuration service is enabled and any of the following conditions occur, the plug-in configuration file is automatically generated:

   - When the Web server is created or saved.
   - When an application is installed.
   - When an application is uninstalled.
   - When the virtual host definition is updated

   Generating or regenerating the configuration file might take a while to complete. After it finishes, all objects in the administrative cell use their newest settings, which the Web server can access. If the Application Server is on the same physical machine as the Web server, the regeneration usually takes about 30 to 60 seconds to complete. The regeneration takes longer if they are not both on the same machine.

   Important: When the plug-in configuration file is first generated, it does not include admin_host on the list of virtual hosts. ″Allowing Web servers to access the administrative console″ in the information center describes how to add it to the list.

   To use the administrative console:

   a. Select **Servers > Web Servers >** *webserver* **> plug-in properties**.
   b. Select **Automatically generate plug-in configuration file** or click on one or more of the following topics to manually configure the `plugin-cfg.xml` file:
      - Caching
      - Request and response
      - Request routing
      - Service

      Web server plug-in configuration properties maps each property to one of these topics.
   c. Click **OK**.
   d. You might need to stop the application server and then start the application server again to enable the Web server to locate the `plugin-cfg.xml` file.

5. If you want to use Secure-socket layer (SSL) with this configuration, use the plug-in’s installation wizard to install the appropriate GSKIT installation image file on your workstation. See “Configuring the Web server plug-in for Secure Sockets Layer” on page 1693 for information on how to configure GSKIT.

6. If you want to enable the Web server plug-in to use private headers, define an SSL configuration repertoire that defines a trust file. Then in the administrative console, select **Application servers > server1 > Web Container Settings > Web Container Transport Chains >** *transport_chain* > **SSL Inbound Channel (SSL_2)** and specify this repertoire for that transport chain. If you try to use private headers without setting up an SSL configuration repertoire that does not include a trust file definition, the private headers will be ignored. If the private headers are ignored, the application server might not locate the requested application.

   After you enable the use of private headers, the transport chain’s SSL inbound channel trusts all private headers it receives. Therefore, you must ensure that all paths to the transport chain’s SSL inbound channel are trusted.

7. Tune your Web server with Web server tuning parameters.

8. Propagate the plug-in configuration. The plug-in configuration file (plugin-cfg.xml) is automatically propagated to the Web server if the Web server plug-in configuration service is enabled, and one of the following is true:

- The Web server is a local Web server. (It are located on the same machine as an application server.)
- The Web server is a remote IBM HTTP Server (IHS) Version 6.0 that has a running IHS Administrative server.

If neither of these conditions is true, the plugin-cfg.xml file must be manually copied to the remote Web server's installation location.

The configuration is complete. To activate the configuration, stop and restart the Web server. If you encounter problems restarting your Web server, check the http_plugin.log file for information on what portion of the plugin-cfg.xml file contains an error. The log file states the line number on which the error occurred along with other details that might help you diagnose why the Web server did not start. You can then use the administrative console to update the plugin-cfg.xml file.

If applications are infrequently installed or uninstalled, which is usually the situation in a production environment, or if you can tolerate the performance impact of generating and distributing the plug-in configuration file each time any of the previously listed actions occur, you should consider enabling this service.

If you are making a series of simultaneous changes, like installing numerous applications, you might want the configuration service disabled until after you make the last change. The Web server plug-in configuration service is enabled by default. To disable this service, in the administrative console click elect **Servers** > **Application Servers** > *server_name* > **Administration Services** >**Web server plug-in configuration service** and then unselect the Enable automated Web server configuration processing option.

Tip: If your installation uses a firewall, make sure you configure the Web server plug-in to use a port that has been opened. (See your security administrator for information on how to obtain an open port.

Web server plug-in properties settings

Use this page to view or change the settings of a Web server plug-in configuration file. The plug-in configuration file, `plugin_cfg.xml`, provides properties for establishing communication between the Web server and the Application Server.

To view this administrative console page, click **Servers > Web Servers >***Web_server_name* **Plug-in Properties**.

On the **Configuration** tab, you can edit fields. On the **Runtime** tab, you can look at read-only information.

The **Runtime** tab is available only when this Web server has accessed applications running on application servers and there is an `http_plugin.log` file.

Plug-in log file name

Specifies the fully qualified path to the log file to which the plug-in will write error messages. The default file path is *plugin_install_root*/`logs`/*web_server_name*/`http_plugin.log` .

If the file does not exist then it will be created. If the file already exists, it will be opened in append mode and the previous plug-in log messages will remain.

This field corresponds to the RequestMetrics loggingEnabled element in the plugin-cfg.xml file.

Data type	String
Default for Linux and UNIX platforms	*plugin_install_root*/`logs`/*web_server_name*/`http_plugin.log`
Default for Windows platforms	*plugin_install_root*/`logs`/*web_server_name*/`http_plugin.log`

Plug-in installation location

Specifies the fully qualified path to where the plug-in configuration file is installed.

Data type	String
Default	The default value is the installation root directory.

If you select a Web server plug-in during installation, the installer program configures the Web server to identify the location of the `plugin-cfg.xml` file, if possible. The Web server is considered installed on a local machine if it is on the same machine as the application server. It is considered installed on a remote machine if the Web server and the application server are on different machines.

- If the Web server is installed on a remote machine, the plug-in configuration file, by default, will be installed in the *plugin_install_root*/config/*web_server_name* directory.
- If the Web server is installed on a local standalone machine, the plug-in configuration file, by default will be installed in the *profile_install_root*/config/cells/*cell_name*/nodes/*web_server_name _node*/servers/*web_server_name* directory.

The installer program adds a directive to the Web server configuration that specifies the location of the `plugin-cfg.xml` file.

For remote Web servers, you must copy the file from the local directory where the Application Server is installed to the remote machine. This is known as propagating the plug-in configuration file. If you are using an IBM HTTP Server (IHS) V6 for your Web server, WebSphere Application Server can automatically propagate the plug-in configuration file for you to remote machines provided there is a working HTTP transport mechanism to propagate the file.

Plug-in configuration file name

Specifies the file name of the configuration file for the plug-in. The Application Server generates the `plugin-cfg.xml` file by default. The configuration file identifies applications, Application Servers, clusters, and HTTP ports for the Web server. The Web server uses the file to access deployed applications on various Application Servers.

Data type	String
Default	`plugin-cfg.xml`

If you select a plug-in during installation, the installer program configures the Web server to identify the location and name of the `plugin-cfg.xml` file, if possible.

You can change the name of the plug-in configuration file. However, if you do change the file name, you must also change the Web server configuration to point to the new plug-in configuration file.

Automatically generate plug-in configuration file

To automatically generate a plug-in configuration file to a remote Web server:

- This field must be checked.
- The plug-in configuration service must be enabled

When the plug-in configuration service is enabled, a plug-in configuration file is automatically generated for a Web server whenever:

- The WebSphere Application Server administrator defines new Web server.
- An application is deployed to an Application Server.
- An application is uninstalled.
- A virtual host definition is updated and saved.

Clear the check box if you want to manually generate a plug-in configuration file for this Web server.

Important: When the plug-in configuration file is generated, it does not include admin_host on the list of virtual hosts. ″Allowing Web servers to access the administrative console″ in the information center describes how to add it to the list.

Automatically propagate plug-in configuration file

To automatically propagate a copy of a changed plug-in configuration file to a Web server:

- This field must be checked.
- The plug-in configuration service must be enabled
- A WebSphere Application Server node agent must be on the node that hosts the Web server associated with the changed plug-in configuration file.

Note: The plug-in configuration file can only be automatically propagated to a remote Web server if that Web server is an IHS V6.0 Web server and its administration server is running.

Ignore DNS failures during Web server startup

Specifies whether the plug-in ignores DNS failures within a configuration when starting.

This field corresponds to the IgnoreDNSFuilures element in the plugin-cfg.xml file.

When set to **true**, the plug-in ignores DNS failures within a configuration and starts successfully if at least one server in each ServerCluster is able to resolve the host name. Any server for which the host name can not be resolved is marked **unavailable** for the life of the configuration. No attempts to resolve the host name are made later on during the routing of requests. If a DNS failure occurs, a log message is written to the plug-in log file and the plug-in initialization continues rather than causing the Web server not to start. When **false** is specified, DNS failures cause the Web server not to start.

Data type	String
Default	false

Refresh configuration interval

Specifies the time interval, in seconds, at which the plug-in should check the configuration file to see if updates or changes have occurred. The plug-in checks the file for any modifications that have occurred since the last time the plug-in configuration was loaded.

In a development environment in which changes are frequent, a lower setting than the default setting of 60 seconds is preferable. In production, a higher value than the default is preferable because updates to the configuration will not occur so often. If the plug-in reload fails for some reason, a message is written to the plug-in log file and the previous configuration is used until the plug-in configuration file successfully reloads. If you are not seeing the changes you made to your plug-in configuration, check the plug-in log file for indications of the problem.

Data type	Integer
Default	60 seconds.

Plug-in logging

Specifies the location and name of the `http_plugin.log` file. Also specifies the scope of messages in the log.

This field corresponds to the RequestMetrics traceLevel element in the plugin-cfg.xml file.

The log describes the location and level of log messages that are written by the plug-in. If a log is not specified within the configuration file, then, in some cases, log messages are written to the Web server error log.

On a distributed platform, if the log file does not exist then it will be created. If the log file already exists, it will be opened in append mode and the previous plug-in log messages will remain.

Log file name - The fully qualified path to the log file to which the plug-in will write error messages.

Data type	String
Default	*plugin_install_root*/logs/*web_server_name*/http_plugin.log Specify the file path of the http_plugin.log file.

Log level- The level of detail of the log messages that the plug-in should write to the log. You can specify one of the following values for this attribute:

- Trace. All of the steps in the request process are logged in detail.
- Stats. The server selected for each request and other load balancing information relating to request handling is logged.
- Warn. All warning and error messages resulting from abnormal request processing are logged.
- Error. Only error messages resulting from abnormal request processing are logged.

If a Log level is not specified, the default value **Error** is used.

Be careful when setting the level to **Trace**. A lot of messages are logged at this level which can cause the disk space/file system to fill up very quickly. A **Trace** setting should never be used in a normally functioning environment as it adversely affects performance.

Data type	String
Default	Error

Web server plug-in request and response optimization properties settings

Use this page to view or change the request and response optimization properties for a Web server plug-in.

To view this administrative console page, click **Servers > Web Servers >***web_server_name* **Plug-in Properties > Request and Response**.

On the **Configuration** tab, you can edit fields. On the **Runtime** tab, you can look at read-only information.

The **Runtime** tab is available only when this Web server has accessed applications running on application servers and there is an http_plugin.log file.

Maximum chunk size used when reading the response body:

Specifies the maximum chunk size the plug-in can use when reading the response body.

This field corresponds to the ResponseChunkSize element in the plugin-cfg.xml file.

The plug-in reads the response body in 64K chunks until all of the response data is read. This approach causes a performance problem for requests whose response body contains large amounts of data.

If the content length of the response body is unknown, the values specified for this property is used as the size of the buffer that is allocated. The response body is then read in this size chunks, until the entire body is read. If the content length is known, then a buffer size of either the content length or the specified size (whichever is less) is used to read the response body.

Data type	Integer

Default	64 kilobytes Specify the size in kilobytes (1024 byte blocks).

Enable Nagle algorithm for connections to the Application Server:

When checked, the Nagle algorithm is enabled for connections between the plug-in and the Application Server.

This field corresponds to the ASDisableNagle element in the plugin-cfg.xml file.

The Nagle algorithm is named after engineer John Nagle, who invented this standard part of the transmission control protocol/internet protocol (TCP/IP). The algorithm reduces network overhead by adding a transmission delay (usually 20 milliseconds) to a small packet, which lets other small packets arrive and be included in the transmission. Because communications has an associated cost that is not as dependent on packet size as it is on frequency of transmission, this algorithm potentially reduces overhead with a more efficient number of transmissions.

Clear the check box to disable the Nagle algorithm.

Enable Nagle Algorithm for the IIS Web Server:

When checked, the Nagle algorithm is used for connections from the Microsoft Internet Informations Services (IIS) Web Server to the Application Server.

This field corresponds to the IHSDisableNagle element in the plugin-cfg.xml file. It only appears if you are using the Microsoft Internet Informations Services (IIS) Web server. Clear the check box to disable the Nagle algorithm for this connection.

Chunk response to the client:

When checked, responses to the client are broken into chunks if a `Transfer-Encoding : Chunked` response header is present in the response.

This field corresponds to the ChunkedResponse element in the plugin-cfg.xml file. It only appears if you are using a Microsoft Internet Informations Services (IIS) Web Server, a Java System Web server, or a Domino Web server. The IHS Web server automatically handles breaking the response into chunks to send to the client.

Clear the check box to if you do not want responses broken into chunks.

Accept content for all requests: This field corresponds to the AcceptAllContent element in the plugin-cfg.xml file.

When selected, users can include content in POST, PUT, GET, and HEAD requests when a Content-Length or Transfer-encoding header is contained in the request header.

Virtual host matching:

When selected, virtual host mapping is performed by physically using the port number for which the request was received.

This field corresponds to the VHostMatchingCompat element in the plugin-cfg.xml file.

If this field is not selected, matching is done logically using the port number contained in the host header.

Use the radio buttons to make your physical or logical port selection.

Application server port preference:

Specifies which port number the Application Server should use to build URI's for a sendRedirect.

This field corresponds to the AppServerPortPreference element in the plugin-cfg.xml file.

Specify:
- `webserverPort` if the port number from the host header of the HTTP request coming in is to be used.
- `hostHeader` if the port number on which the Web server received the request is to be used.

The default is `webserverPort`.

Priority used by the IIS Web server when loading the plug-in configuration file:

Specifies the priority in which the Microsoft Internet Informations Services (IIS) Web server loads the WebSphere Web server plug-in.

This field corresponds to the IISPluginPriority element in the plugin-cfg.xml file. It only appears if you are using the IIS Web server. Because the IIS Web server uses this value during startup, the Web server must be restarted before a change to this field takes effect.

Select one of the following priorities:
- High
- Medium
- Low

The default value of **High** ensures that all requests are handled by the Web server plug-in before they are handled by any other filter/extensions. If problems occur while using a priority of **Medium** or **Low**, you will have to rearrange the order or change the priority of the interfering filter/extension.

Web server plug-in caching properties settings

Use this page to view or change the caching properties for a Web server plug-in.

To view this administrative console page, click **Servers > Web Servers >***Web_server_name* **Plug-in Properties > Caching Properties**.

On the **Configuration** tab, you can edit fields. On the **Runtime** tab, you can look at read-only information.

The **Runtime** tab is available only when this Web server has accessed applications running on application servers and there is an `http_plugin.log` file.

Enable Edge Side Include (ESI) processing to cache the responses:

Specifies whether to enable Edge Side Include processing to cache the responses.

This field corresponds to the esiEnable element in the plugin-cfg.xml file.

When selected, Edge Side Include (ESI) processing is used to cache responses. If ESI processing is disabled for the plug-in, the other ESI plug-in properties are ignored. Clear the checkbox to disable Edge Side Include processing.

Enable invalidation monitor to receive notifications:

When checked, the ESI processor receives invalidations from the Application Server.

This field corresponds to the ESIInvalidationMonitor element in the plugin-cfg.xml file. It is ignored if Edge Side Include (ESI) processing is not enabled for the plug-in.

Maximum cache size:

Specifies, in 1K byte units, the maximum size of the cache. The default maximum size of the cache is 1024K bytes (1 megabyte). If the cache is full, the first entry to be evicted from the cache is the entry that is closest its expiration time.

This field corresponds to the esiMaxCacheSize element in the plugin-cfg.xml file.

Data type	Integer
Default	1024 kilobytes Specify the size in kilobytes (1024 byte blocks).

Web server plug-in request routing properties settings

Use this page to view or change the request routing properties for a Web server plug-in.

To view this administrative console page, click **Servers > Web Servers >***Web_server_name* **Plug-in Properties > Plug-in** *server_cluster_name* **Properties**.

On the **Configuration** tab, you can edit fields. On the **Runtime** tab, you can look at read-only information.

The **Runtime** tab is available only when this Web server has accessed applications running on application servers and there is an `http_plugin.log` file.

Load balancing option:

Specifies the load balancing option that the plug-in uses in sending requests to the various application servers associated with that Web server.

This field corresponds to the LoadBalanceWeight element in the plugin-cfg.xml file.

Select the appropriate load balancing option:

- Round robin
- Random

The Round Robin implementation has a random starting point. The first application server is picked randomly. Round Robin is then used to pick application servers from that point forward. This implementation ensures that in multiple process based Web servers, all of the processes don't start up by sending the first request to the same Application Server.

The default load balancing type is Round Robin.

Retry interval:

Specifies the length of time, in seconds, that should elapse from the time an application server is marked down to the time that the plug-in retries a connection.

This field corresponds to the ServerWaitforContinue element in the plugin-cfg.xml file.

Data type	Integer
Default	60 seconds

Maximum size of request content:

Select whether there is a limit on the size of request content. If limited, this field also specifies the maximum number of bytes of request content allowed in order for the plug-in to attempt to send the request to an application server.

This field corresponds to the PostSizeLimit element in the plugin-cfg.xml file.

When a limit is set, the plug-in fails any request that is received that is greater than the specified limit.

Select whether to limit the size of request content:
- No limit
- Set limit

If **Set limit** is selected, specify a limit size.

Data type	Integer
Default	-1, which indicates there is no limit for the post size. Specify the size in kilobytes (1024 byte blocks).

Remove special headers:

When checked, the plug-in will remove any headers from incoming requests before adding the headers the plug-in is supposed to add before forwarding the request to an application server.

This field corresponds to the RemoveSpecialHeaders element in the plugin-cfg.xml file.

The plug-in adds special headers to the request before it is forwarded to the application server. These headers store information about the request that will need to be used by the application. Not removing the headers from incoming requests introduces a potential security exposure.

Clear the check box to retain special headers.

Clone separator change:

When checked, the plug-in expects the plus character (+) as the clone separator.

This field corresponds to the ServerCloneID element in the plugin-cfg.xml file.

Some pervasive devices cannot handle the colon character (:) used to separate clone IDs in conjunction with session affinity. If this field is checked, you must also change the configurations of the associated application servers such that the application servers separates clone IDs with the plus character as well.

Clear the checkbox to use the colon character to separate clone IDs.

Web server plug-in custom properties

If you are using a Web server plug-in, you can add the following custom property to the configuration settings for that plug-in.

To add a custom property:
1. In the administrative console, click In the administrative console, click **Servers > Web Servers >** *Web_server_name* **> Plug-in properties > Custom properties > New**

2. Under **General Properties** specify the name of the custom property in the Name field and a value for this property in the Value field. You can also specify a description of this property in the Description field.
3. Click **Apply** or **OK**.
4. Click **Save** to save your configuration changes.
5. Restart the server.

Following is a list of Web server plug-in custom properties that are provided with the Application Server. These properties are not shown on the properties settings pages for the plug-in.

StashfileLocation

Use this element to set a value for the stashfile initialization parameter.

Data type	String

KeyringLocation

Use this element to set a value for the keyring initialization parameter.

Data type	String

Web server plug-in configuration service properties settings

Use this page to view or change the configuration settings for the Web server plug-in configuration service.

To view this administrative console page, click **Application Servers >***server_name* > **Administration Services** > **Web server plug-in configuration service**.

On the **Configuration** tab, you can edit fields. On the **Runtime** tab, you can look at read-only information.

Enable automated Web server configuration processing

When selected, the Web server plug-in configuration service automatically generates the plug-in configuration file whenever the Web server environment changes. For example, the plug-in configuration file is regenerated whenever one of the following activities occurs:

- A new application is deployed on an associated application server.
- The Web server definition is saved.
- An application is removed from an associated application server.
- A new virtual host is defined.

Whenever a virtual host definition is updated, the plug-in configuration file is automatically regenerated for all of the Web servers.

Application Server property settings for a Web server plug-in

Use this page to view or change application server settings for a Web server plug-in.

To view this administrative console page, click **Application Servers >***server_name* **> Web Server Plug-in**.

On the **Configuration** tab, you can edit fields. On the **Runtime** tab, you can look at read-only information.

The **Runtime** tab is available only when this Web server has accessed applications running on application servers and there is an `http_plugin.log` file.

Server role

Specifies the role this application server is assigned.

Select `Primary` to add this application server to the list of primary application servers. The plug-in initially attempts to route requests to the application servers on this list.

Select `Backup` to add this application server to the list of backup application servers. The plug-in does not load balance across the backup application servers. A backup server is only used if a primary server is not available. When the plug-in determines that a backup application server is required, it goes through the list of backup servers, in order, until no servers are left in the list or until a request is successfully sent and a response received from one of the servers on this list.

Connect timeout

Specifies whether or not there is a limited amount of time the Application Server will maintain a connection with the Web server.

You can select either **No timeout** or **Set timeout**. If you select **Set timeout** you, must specify, in seconds, the length of time a connection with the Web server is to be maintained.

This property enables the plug-in to perform non-blocking connections with the application server. Non-blocking connections are beneficial when the plug-in is unable to contact the destination to determine whether or not the port is available. If no value is specified for this property, the plug-in performs a blocking connect in which the plug-in sits until an operating system times out (which could be as long as 2 minutes depending on the platform) and allows the plug-in to mark the server `unavailable`.

A value of 0 causes the plug-in to perform a blocking connect. A value greater than 0 specifies the number of seconds you want the plug-in to wait for a successful connection. If a connection does not occur after that time interval, the plug-in marks the server unavailable and fails over to another application server defined for the requested application.

Data type	Integer

Maximum number of connections that can be handled by the Application Server

Specifies the maximum number of pending connections to an Application Server that can be flowing through a Web server process at any point in time.

This field corresponds to the ServerMaxConnections element in the plugin-cfg.xml file.

You can select either **No limit** or **Set limit**. If you select **Set limit** you, must specify the maximum number of connections that can exist between the Web server and the Application Server at any given point in time.

If this attribute is set to either zero or -1, there is no limit to the number of pending connections to the Application Servers.

Data type	Integer
Default	-1

Use extended handshake to check whether Application Server is running

When selected, the Web server plug-in will use an extended handshake to check whether or not the Application Server is running.

This field corresponds to the ServerExtendedHandshake element in the plugin-cfg.xml file.

Select this property if a proxy firewall is between the plug-in and the application server.

The plug-in marks a server as down when the connect() fails. However, when a proxy firewall is in between the plug-in and the application server, the connect() will succeed, even though the back end application server is down. This causes the plug-in to not failover correctly to other application servers.

If the plug-in performs some handshaking with the application server to ensure that it is started before it sends a request it can failover to another application server if it detects that the application server with which it is attempting to perform a handshake is down.

Send the header ″100 Continue″ before sending the request content

This field corresponds to the WaitForContinue element in the plugin-cfg.xml file.

When selected, the Web server plug-in will send the header ″100 Continue″ to the Application Server before it sends the request content.

Web server plug-in configuration properties

The following table indicates which panel in the administrative console you need to use to manually configure a Web server plug-in property.

Table 5. Web server plug-in configuration properties

Administrative console panel	Field name	Configuration property name
In the administrative console, click **Servers** > **Web Servers** >*Web_server_name* > **Plug-in properties**	Refresh configuration interval	RefreshInterval
In the administrative console, click **Servers** > **Web Servers** > *Web_server_name* > **Plug-in properties**	Plug-in log file name	Log->name
In the administrative console, click **Servers** > **Web Servers** > *Web_server_name* > **Plug-in properties**	Plug-in logging	Log->LogLevel
In the administrative console, click **Servers** > **Web Servers** > *Web_server_name* > **Plug-in properties**	Ignore DNS failures during Web server startup	IgnoreDNSFailures
In the administrative console, click **Servers** > **Web Servers** > *Web_server_name* > **Plug-in properties** > **Custom properties** > **New**	KeyringLocation	Keyring
In the administrative console, click **Servers** > **Web Servers** > *Web_server_name* > **Plug-in properties** > **Custom properties** > **New**	StashfileLocation	Stashfile
In the administrative console, click **Servers** > **Web Servers** > *Web_server_name* > **Plug-in properties** > **Request routing**	Load balancing option	LoadBalance
In the administrative console, click **Servers** > **Web Servers** > *Web_server_name* > **Plug-in properties** > **Request Routing**	Clone separator change	CloneSeparatorChange

Table 5. Web server plug-in configuration properties (continued)

In the administrative console, click **Servers > Web Servers >** *Web_server_name* **> Plug-in properties > Request Routing**	Retry interval	RetryInterval
In the administrative console, click **Servers >***Web server_name* **> Plug-in properties > Request routing**	Maximum size of request content	PostSizeLimit
In the administrative console, click **Servers > Web Servers >** *Web_server_name* **> Plug-in properties > Request routing**	Remove special headers	RemoveSpecialHeaders
In the administrative console, click **Application Servers >***server_name* **> Web server plug-in properties**	Server role	PrimaryServers and BackupServers list
In the administrative console, click **Application Servers >***server_name* **> Web server plug-in properties**	Connect timeout	Server ConnectTimeout
In the administrative console, click **Application Servers >***server_name* **> Web server plug-in properties**	Use extended handshake to check whether Application Server is running	Server Extended Handshake
In the administrative console, click **Application Servers >***server_name* **> Web server plug-in properties**	Send the header ″100 Continue″ before sending the request content	WaitForContinue
In the administrative console, click **Application Servers >***server_name* **> Web server plug-in properties**	Maximum number of connections that can be handled by the Application Server	Server MaxConnections
In the administrative console, click **Servers > Web Servers >** *Web_server_name* **> Plug-in properties > Request and Response**	Application server port preference	AppServerPortPreference
In the administrative console, click **Servers > Web Servers >** *Web_server_name* **> Plug-in properties > Request and Response**	Enable Nagle algorithm for connections to the Application Server	ASDisableNagle
In the administrative console, click **Servers > Web Servers >** *Web_server_name* **> Plug-in properties > Request and Response**	Enable Nagle Algorithm for the IIS Web Server	IISDisableNagle
In the administrative console, click **Servers > Web Servers >** *Web_server_name* **> Plug-in properties > Request and Response**	Virtual host matching	VHostMatchingCompat
In the administrative console, click **Servers > Web Servers >** *Web_server_name* **> Plug-in properties > Request and Response**	Maximum chunk size used when reading the response body	ResponseChunkSize

Table 5. Web server plug-in configuration properties (continued)

In the administrative console, click **Servers** > **Web Servers** > *Web_server_name* > **Plug-in properties** > **Request and Response**	Accept content for all requests	AcceptAllContent
In the administrative console, click **Servers** > **Web Servers** > *Web_server_name* > **Plug-in properties** > **Request and Response**	Chunk response to the client	ChunkedResponse
In the administrative console, click **Servers** > **Web Servers** > *Web_server_name* > **Plug-in properties** > **Request and Response**	Priority used by the IIS Web server when loading the plug-in configuration file	IISPluginPriority
In the administrative console, click **Servers** > **Web Servers** > *Web_server_name* > **Plug-in properties** > **Caching**	Enable Edge Side Include (ESI) processing to cache the responses	ESIEnable
In the administrative console, click **Servers** > **Web Servers** > *Web_server_name* > **Plug-in properties** > **Caching**	Maximum cache size	ESIMaxCacheSize
In the administrative console, click **Servers** > **Web Servers** > *Web_server_name* > **Plug-in properties** > **Caching**	Enable invalidation monitor to receive notifications	ESIInvalidationMonitor

Web server plug-in connections

The WebSphere Application Server Web server plug-ins are used to establish and maintain persistent HTTP and HTTPS connections to Application Servers .

When the plug-in is ready to send a request to the application server, it first checks its connection pool for existing connections. If an existing connection is available the plug-in checks its connection status. If the status is still good, the plug-in uses that connection to send the request. If a connection does not exist, the plug-in creates one. If a connection exists but has been closed by the application server, the plug-in closes that connection and opens a new one.

After a connection is established between a plug-in and an application server, it will not be closed unless the application server closes it for one of the following reasons:

- If the **Use Keep-Alive** property is selected and the time limit specified on the **Read timeout** or **Write timeout** property for the HTTP inbound channel has expired.
- The maximum number of persistent requests which can be processed on an HTTP inbound channel has been exceeded. (This number is set using the HTTP inbound channel’s **Maximum persistent requests** property.)
- The Application Server is shutting down.

Even if the application server closes a connection, the plug-in will not know that it has been closed until it tries to use it again. The connection will be closed if one of the following events occur:

- The plug-in receives a new HTTP request and tries to reuse the existing connection.

- The number of **httpd** processes drop because the Web server is not receiving any new HTTP requests. (For the IHS Web server, the number of **httpd** processes that are kept alive depends on the value specified on the Web server's MinSpareServers directive.)
- The Web server is stopped and all **httpd** processes are terminated, and their corresponding sockets are closed.

Note: Sometimes, if a heavy request load is stopped or decreased abruptly on a particular application server, a lot of the plug-in's connections to that application server will be in CLOSE_WAIT state. Because these connections will be closed the first time the plug-in tries to reuse them, having a large number of connections in CLOSE-WAIT state should not affect performance

Web server plug-in remote user information processing

You can configure your Web server with a third-party authentication module and then configure the Web server plug-in to route requests to the Application Server. If an application calls the getRemoteUser() method, it relies on a private HTTP header that contains the remote user information and is parsed by the plug-in. The plug-in sets the private HTTP header value whenever a Web server authentication module populates the remote user in the Web server data structure. If the private HTTP header value is not set, the application's call to getRemoteUser() returns a null value.

- In the case of an Apache and IBM HTTP Server (IHS) Web server, the plug-in builds the private header from the information contained in the associated request record.
- In the case of a Sun One Web server, the plug-in builds the private header from the information contained in the **auth_user** property associated with the request. The private header is usually set to the name of the local HTTP user of the Web browser, if HTTP access authorization is activated for the URL.
- In the case of a Domino Web server, the plug-in builds the private header from the information contained in the **REMOTE_USER** environment variable. The plug-in sets this variable to **anonymous** for users who have not logged in and to the *username* for users who are logged into the application.
- In the case of an Internet Information Services (IIS) Web server, the plug-in builds the private header from the information contained in the **REMOTE_USER** environment variable. The plug-in sets this variable to the name of the user as it is derived from the authorization header sent by the client.

If the private header is not being set in the Sun One, IIS, or Domino Web server plug-in, make sure the request record includes information about the user requesting the data.

Note: If an application's call to getRemoteUser() returns a null value, or if the correct remote user information is not being added to the Web server plug-in's data structure, make sure the remote user parameter within the WebAgent is still set to **YES**. (Sometimes this parameter gets set to **NO** when service is applied.)

Web server plug-ins

Web server plug-ins enable the Web server to communicate requests for dynamic content, such as servlets, to the application server. A Web server plug-in is associated with each Web server definition. The configuration file (plugin-cfg.xml) that is generated for each plug-in is based on the applications that are routed through the associated Web server.

A Web server plug-in is used to forward HTTP requests from a supported Web server to an application server. Using a Web server plug-in to provide communication between a Web server and an application server has the following advantages:

- XML-based configuration file
- Standard protocol recognized by firewall products
- Security using HTTPS, replacing proprietary Open Servlet Engine (OSE) over Secure Sockets Layer (SSL)

Each of the supported distributed platform Web server plug-ins run on a number of operating systems. See Supported Hardware and Software at `http://www.ibm.com/software/webservers/appserv/doc/latest/prereq.html`for the product for the most current information about supported Web servers.

Checking your IBM HTTP Server version

At times, you might need to determine the version of your IBM HTTP Server installation.

1. Change directory to the installation root of the Web server. For example, this is /opt/IBMIHS on a Solaris machine.
2. Find the subdirectory that contains apache.exe (on a Windows platforms) or apachectl (on a UNIX-based platforms, such as z/OS, Solaris, Linux, and HP-UX)
3. On a Windows platform, issue:

   ```
   apache.exe -V
   ```
4. On a UNIX-based platforms issue:

   ```
   ./apachectl -V 4
   ```

The version is shown in the ″Server version:″ field and will looks something like the following:

```
Server version: IBM_HTTP_Server/2.0.47 Apache/2.0.47
Server built: July 2 2004 20:38:36
Server's Module Magic Number: 20020903:4.
```

Web server tuning parameters

WebSphere Application Server provides plug-ins for several Web server brands and versions. Each Web server operating system combination has specific tuning parameters that affect the application performance.

- **IBM HTTP Server**

  The IBM HTTP Server V6.0 is a multi-process, multi-threaded server.

  - **Access logs**
    - **Description:** Collects all incoming HTTP requests. Logging degrades performance because IO operation overhead causes logs to grow significantly in a short time.
    - **How to view or set:**
      1. Open the IBM HTTP Server `httpd.conf` file, located in the directory *IBM_HTTP_Server_root_directory*/conf.
      2. Search for a line with the text **CustomLog**.
      3. Comment out this line by placing # in front of the line.
      4. Save and close the `httpd.conf` file.
      5. Stop and restart the IBM HTTP Server.
    - **Default value:** Logging of every incoming HTTP request is enabled.
    - **Recommended value:** Disable the access logs.
  - **MaxClients**
    - **Description:** The MaxClients directive controls the maximum number of simultaneous connections or users that the web server can service at any one time. If, at peak usage, your web server needs to support 200 active users at once, you should set MaxClients to 220 (200 plus an extra 10% for load growth). Setting MaxClients too low could cause some users to believe the web server is not responding. You should have sufficient RAM in your web server machines to support each connected client. For IBM HTTP Server V6.0 on UNIX, you should allocate around 1.5MB MaxClients of RAM for use by the IBM HTTP Server. For IBM HTTP Server V6.0 on Windows, you should allocate around 300KB MaxClients of RAM for use by the IBM HTTP Server. Some third party modules can significantly increase the amount of RAM used per connected client.
    - **How to view or set:** Edit the MaxClients directive in the IBM HTTP Server `httpd.conf` file, located in the directory *IBM_HTTP_Server_root_directory*/conf.
    - **Default value:** 150

    - **Recommended value:** The maximum number of users normally simultaneously connected to your web server, plus an additional 10% for buffer. Note: The KeepAliveTimeout setting can affect how long a user is connected to the webserver.
  - **MinSpareServers, MaxSpareServers, and StartServers**
    - **Description:** Pre-allocates and maintains the specified number of processes so that few processes are created and destroyed as the load approaches the specified number of processes. Specifying similar values reduces the CPU usage for creating and destroying HTTPD processes. Adjust this parameter if the time waiting for IBM HTTP Server to start more servers, so that it can handle HTTP requests, is not acceptable.
    - **How to view or set:** Edit the MinSpareServers, MaxSpareServers and StartServers directives in the `httpd.conf` file located in the *IBM_HTTP_Server_root_directory*/conf directory.
    - **Default value:** MinSpareServers 5, MaxSpareServers 10, StartServers 5
    - **Recommended value:** For optimum performance, specify the same value for the MinSpareServers and the StartServers parameters. If MaxSpareServers is set to less than MinSpareServers, IBM HTTP Server resets MaxSpareServer=MinSpareServer+1. Setting the StartServers too high can cause swapping if memory is not sufficient, degrading performance.
  - **ListenBackLog**
    - **Description:** Sets the length of a pending connections queue. When several clients request connections to the IBM HTTP Server, and all threads used, a queue exists to hold additional client requests. However, if you use the default Fast Response Cache Accelerator (FRCA) feature of IBM HTTP Server V6.0 on Windows, the ListenBackLog directive is not used since FRCA has its own internal queue.
    - **How to view or set:** For non-FRCA: Edit the IBM HTTP Server `httpd.conf` file. Then, add or view the ListenBackLog directive.
    - **Default value:** For HTTP Server V6.0: 1024 with FRCA enabled, 511 with FRCA disabled
    - **Recommended value:** Use the defaults.
- **IBM HTTP Server - Linux**
  - **MaxRequestsPerChild**
    - **Description:** Sets the limit on the number of requests that an individual child server process handles. After the number of requests reaches the value set for the MaxRequestsPerChild parameter, the child process dies. Adjust this parameter if destroying and creating child processes is degrading your Web server performance.
    - **How to view or set:**
      1. Edit the IBM HTTP server `httpd.conf` file located in the *IBM_HTTP_Server_root_directory*/conf directory.
      2. Change the value of the parameter.
      3. Save the changes and restart the IBM HTTP server.
    - **Default value:** 500
    - **Recommended value:** Should normally be set to 0. Non-zero settings can be useful if child memory usage is observed to steadily increase over time. Memory leaks have occasionally been observed in third party modules and various OS runtime libraries used by the IBM HTTP Server.
- **IBM HTTP Server - Windows 2000 and Windows 2003**
  - **ThreadsPerChild**
    - **Description:** Sets the number of concurrent threads running at any one time within the IBM HTTP Server.
    - **How to view or set:** Edit the IBM HTTP Server file `httpd.conf` file located in the directory *IBM_HTTP_Server_root_directory*/conf. Change the value of the parameter. Save the changes and restart the IBM HTTP Server.

      There are two ways to find how many threads are used under load:
      1. Use the Windows 2000 and Windows 2003 Performance Monitor under the desktop Start menu:
         a. Right-click the status bar on your desktop. Click **Task Manager**.
         b. Select the **Processes** tab.
         c. Click **View** > **Select Columns**.
         d. Select **Thread Count**.
         e. Click **OK**.

The **Processes** tab shows the number threads for each process under the column name **Threads**, including Apache.

2. Use the IBM HTTP Server server-status (this choice works on all platforms, not just Windows):
   a. Edit the IBM HTTP Server `httpd.conf` file as follows: Remove the comment character # from the following lines: `#LoadModule status_module`, `modules/ApacheModuleStatus.dll`, `#<Location/server-status>`, `#SetHandler server-status`, and `#</Location>`.
   b. Save the changes and restart the IBM HTTP Server.
   c. In a Web browser, go to the URL: `http://`*`yourhost`*`/server-status`. Alternatively,
   d. Click **Reload** to update status.
   e. (Optional) If the browser supports refresh, go to `http://`*`your_host`*`/server-status?refresh=5` to refresh every five seconds. You will see five requests currently processing 45 idle servers.

- **Default value:** 250 for IBM HTTP Server V6.0.
- **Recommended value:** Set this value to prevent bottlenecks, allowing just enough traffic through to the application server.

– **Web server configuration reload interval**

- **Description:** Tracks a variety of configuration information about WebSphere Application Server resources. The Web server needs to understand some of this information, such as Uniform Resource Identifiers (URIs) pointing to WebSphere Application Server resources. This configuration data is pushed to the Web server through the WebSphere Application Server plug-in at intervals specified by this parameter. Periodic updates add new servlet definitions without having to restart any of the WebSphere Application Server servers. However, the dynamic regeneration of this configuration information is costly in terms of performance. Adjust this parameter in a stable production environment.
- **How to view or set:**Use the Refresh configuration interval Web server plug-in property to change the current setting for this parameter. In the administrative console, click **Servers > Web Servers >***Web_server_name* **> Plug-in properties**.
- **Default value:** The default reload interval is 60 seconds.
- **Recommended value:** Increase the reload interval to a value that represents an acceptable wait time between the servlet update and the Web server update.

For more information about the `plugin-cfg.xml` file see the topic "Web server plug-ins" on page 72.

- **Sun Java System Web server, Enterprise Edition (formerly Sun ONE) - Solaris operating environment**

  The default configuration of the Sun ONE Web server, Enterprise Edition provides a single-process, multi-threaded server.

  – **Active threads**

  - **Description:** Specifies the current number of threads active in the server. After the server reaches the limit set with this parameter, the server stops servicing new connections until it finishes old connections. If this setting is too low, the server can become throttled, resulting in degraded response times. To tell if the Web server is being throttled, consult its perfdump statistics. Look at the following data:
    - **WaitingThreads count:** If WaitingThreads count is getting close to zero, or is zero, the server is not accepting new connections.
    - **BusyThreads count:** If the WaitingThreads count is close to zero, or is zero, BusyThreads is probably very close to its limit.
    - **ActiveThreads count:** If ActiveThreads count is close to its limit, the server is probably limiting itself.
  - **How to view or set:** Use the Maximum number of simultaneous requests parameter in the Enterprise Server Manager interface to control the number of active threads within Sun ONE Web server, Enterprise Edition. This setting corresponds to the RqThrottle parameter in the `magnus.conf` file.
  - **Default value:** 512
  - **Recommended value:** Increase the thread count until the active threads parameters show optimum behavior.

- **Microsoft Internet Information Server (IIS) - Windows NT and Windows 2000**
  - **IIS permission properties**
    - **Description:** The Web server has several properties that dramatically affect the performance of the application server. The default settings are usually acceptable. However, because other products can change the default settings without user knowledge, make sure to check the IIS settings for the Home Directory permissions of the Web server. The permissions should be set to Script and not to Execute. If the permissions are set to Execute, no error messages are returned, but the performance of WebSphere Application Server is decreased.
    - **How to view or set:** To check or change these permissions, perform the following procedure in the Microsoft management console:
      1. Select the Web site (usually default Web site).
      2. Right-click and select the **Properties** option.
      3. Click the **Home Directory** tab. To set the permissions of the Home Directory:
         a. In the **Application** settings, select the **Script** check box in the **Permissions** list and clear the **Execute** check box.
         b. (Optional) Check the permissions of the sePlugin:
            1) Expand the Web server.
            2) Right-click the sePlugin and select **Properties**.
            3) Confirm that the **Execute** permissions are set to **Execute**.
    - **Default value:** Script
    - **Recommended value:** Script
  - **Number of expected hits per day**
    - **Description:** Controls the memory that IIS allocates for connections.
    - **How to view or set:** Using the performance window, set the parameter to More than 100000 in the Web site properties panel of the Microsoft management console.
    - **Default value:** Fewer than 100000
    - **Recommended value:** More than 100000
  - **ListenBackLog parameter**
    - **Description:** Alleviates failed connections under heavy load conditions, if you are using IIS on Windows NT and Windows 2000. Failure typically occurs when you are using more than 100 clients. ListenBackLog increases the number of requests that IIS keeps in its queue. Consider raising this value if you see intermittent `Unable to locate server errors` in the Netscape browser.
    - **How to view or set:**
      1. Use a command prompt to issue the **regedit** command to access the operating system registry.
      2. In the registry window, locate the parameter in the HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\InetInfo\Parameters\ListenBackLog directory.
      3. Right-click the parameter to adjust the setting according to the server load.
    - **Default value:** 25 (decimal)
    - **Recommended value:** You can set the ListenBackLog parameter can be set as high as 200, without negative impact on performance and with an improvement in load handling.

Modifying the WebSphere plug-in to improve performance

You can improve the performance of IBM HTTP Server V6.0 (with the WebSphere Web server plug-in) by modifying the plug-in's RetryInterval configuration. The RetryInterval is the length of time to wait before trying to connect to a server that has been marked temporarily unavailable. Making this change can help the IBM HTTP Server V6.0 to scale higher than 400 users.

The plug-in marks a server temporarily unavailable if the connection to the server fails. Although a default value is 60 seconds, it is recommended that you lower this value in order to increase throughput under heavy load conditions. Lowering the RetryInterval is important for IBM HTTP Server V6.0 on UNIX operating systems that have a single thread per process, or for IBM HTTP Server 2.0 if it is configured to have fewer than 10 threads per process.

How can lowering the RetryInterval affect throughput? If the plug-in attempts to connect to a particular application server while the application server threads are busy handling other connections, which happens under heavy load conditions, the connection times out and the plug-in marks the server temporarily unavailable. If the same plug-in process has other connections open to the same server and a response is received on one of these connections, the server is marked again. However, when you use the IBM HTTP Server V6.0 on a UNIX operating system, there is no other connection since there is only one thread and one concurrent request per plug-in process. Therefore, the plug-in waits for the RetryInterval before attempting to connect to the server again.

Since the application server is not really down, but is busy, requests are typically completed in a small amount of time. The application server threads become available to accept more connections. A large RetryInterval causes application servers that are marked temporarily unavailable, resulting in more consistent application server CPU utilization and a higher sustained throughput.

Note: Although lowering the RetryInterval can improve performance, if all the application servers are running, a low value can have an adverse affect when one of the application servers is down. In this case, each IBM HTTP Server V6.0 process attempts to connect and fail more frequently, resulting in increased latency and decreased overall throughput.

Gskit install images files

The Global Security Kit (GSKit) installation image files for the WebSphere Web server plug-ins are packaged on the CD with the Web server plug-in files. You can download the appropriate GSKIT file to the workstation on which your Web server is running. Use the following table to assist you in selecting the correct GSKIT installation image file.

Operating system	GSKit 7 Installation image file
Windows	No image name
AIX	gskta.rte
HP-UX	gsk7bas
Solaris Operating Environment	gsk7bas
Linux	gsk7bas_7.0.3.1.i386.rpm
Linux390	gsk7bas-7.0.3.1.s390.rpm
LinuxPPC	gsk7bas-7.0.3.1.ppc.rpm

Plug-ins: Resources for learning

See this topic in the V6 Information Center to find links links to relevant supplemental information about Web server plug-ins. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

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

Web server plug-in tuning tips

Balancing workloads:

In a distributed environment, you can limit the number of connections that can be handled by an applications server. To do this, go to the **Servers > Web Servers >** *webserver* **> Plug-in properties** page in the administrative console and select **Set limit** for the **Minimum number of connections that can be handled by the Application Server** field. Then specify in the **Connections** field the maximum number of connections you want to allow. When this maximum number of connections is reached, the plug-in returns an HTTP 503 response code to the client. This code indicates that the server is currently unable to handle the request because it is experiencing a temporary overloading or because maintenance is being performed.

When this maximum number of connections is reached, the plug-in, when establishing connections, automatically skips that application server, and tries the next available application server. If no application servers are available, an HTTP 503 response code will be returned to the client. This code indicates that the server is currently unable to handle the request because it is experiencing a temporary overloading or because maintenance is being performed.

Limiting the number of connections that can be established with an application server works best for Web servers that follow the threading model instead of the process model, and only one process is started.

The IBM HTTP Server V6.0.x follows the threading model. To prevent the IBM HTTP Server from starting more than one process, change the following properties in the Web server configuration file (`httpd.conf`) to the indicated values:

```
ServerLimit          1
ThreadLimit          4000
StartServers         1
MaxClients           1024
MinSpareThreads      1
MaxSpareThreads      1024
ThreadsPerChild      1024
MaxRequestsPerChild  0
```

Windows **Improving performance in a high stress environment:**

If you use the default settings for a Microsoft Windows operating system, you might encounter Web server plug-in performance problems if you are running in a high stress environment. To avoid these problems, consider tuning the the TCP/IP setting for this operating system. Two of the keys setting to tune are TcpTimedWaitDelay and MaxUserPort.

To tune the TcpTimedWaitDelay setting, change the value of the tcp_time_wait_interval parameter from the default value of 240 seconds, to 30 seconds:

1. Locate in the Windows Registry:

   ```
   HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\tcpip\Parameters\TcpTimedWaitDelay
   ```

   If this entry does not exist in your Windows Registry, create it by editing this entry as a new DWORD item.
2. Specify, in seconds, a value between 30 and 300 inclusive for this entry. (It is recommended that you specify a value of 30.)

To tune the MaxUserPort setting:

1. Locate in the Windows Registry:

   ```
   HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\tcpip\Parameters\MaxUserPort
   ```

   If this entry does not exist in your Windows Registry, create it by editing this entry as a new DWORD item.
2. Set the maximum number of ports to a value between 5000 and 65534 ports, inclusive. (It is recommended that you specify a value of 65534,)

See the Microsoft Web site for more information about these settings.

Tuning Web servers

“Web server tuning parameters” on page 73 lists tuning parameters specific to Web servers. The listed parameters may not apply to all of the supported Web servers. Check your Web server documentation before using any of these parameters.

Setting up the administrative architecture

If your system uses administrative services, you can specify settings for the service.

Use the settings page for an administrative service to configure administrative services.

Administration service settings

Use this page to view and change the configuration for an administration service.

To view this administrative console page, click **Servers > Application Servers >***server_name* **> Administration > Administration Services**

Standalone

Specifies whether the server process is a participant in a Network Deployment cell or not. If the box is checked (true), the server does not participate in distributed administration. If the box is unchecked (false), the server participates in the Network Deployment system.

The default value for base WebSphere Application Server installations is true. When addNode runs to incorporate the server into a Network Deployment cell, the value switches to false.

Data type	Boolean
Default	true

Preferred Connector

Specifies the preferred JMX Connector type. Available options, such as SOAPConnector or RMIConnector, are defined using the JMX Connectors page.

Data type	String
Default	SOAP

Extension MBean Providers collection

Use this page to view and change the configuration for JMX extension MBean providers.

You can configure JMX extension MBean providers to be used to extend the existing WebSphere managed resources in the core administrative system. Each MBean provider is a library containing an implementation of a JMX MBean and its MBean XML Descriptor file.

To view this administrative console page, click **Servers > Application Servers >***server_name* **> Administration > Administration Services > Extension MBean Providers**

Name The name used to identify the Extension MBean provider library.

Description
An arbitrary descriptive text for the Extension MBean Provider configuration.

Classpath
The path to the Java archive (JAR) file that contains the Extension MBean provider library. This class path is automatically added to the Application Server class path.

Extension MBean Provider settings:

Use this page to view and change the configuration for a JMX extension MBean provider.

You can configure a library containing an implementation of a JMX MBean, and its MBean XML Descriptor file, to be used to extend the existing WebSphere managed resources in the core administrative system

To view this administrative console page, click **Servers > Application Servers >***server_name* **> Administration > Administration Services > Extension MBean Providers >***provider_library_name*

Classpath: The path to the Java archive (JAR) file that contains the Extension MBean provider library. This class path is automatically added to the Application Server class path. The class loader needs this information to load and parse the Extension MBean XML Descriptor file.

Description: An arbitrary descriptive text for the Extension MBean Provider configuration. Use this field for any text that helps identify or differentiate the provider configuration.

Name: The name used to identify the Extension MBean provider library.

Extension MBean collection

You can configure Java Management Extension (JMX) MBeans to extend the existing WebSphere Application Server managed resources in the administrative console. Use this page to register JMX MBeans. Any MBeans that are listed have already been registered.

To view this administrative console page, click **Servers > Application Servers >***server name* **> Administration > Administration Services > Extension MBean Providers >** *provider library name***> Extension MBean**

DescriptorURI
Specifies the location, relative to the provider class path, where the MBean XML descriptor file is located.

Type Specifies the type to use for registering this MBean. The type must match the type that is declared in the MBean descriptor file.

Extension MBean settings:

Use this page to view and configure Java Management Extension (JMX) MBeans.

To view this administrative console page, click **Servers > Application Servers >***server name***> Administration > Administration Services > Extension MBean Providers >** *provider library name***> Extension MBean >***Extension MBean name*

descriptorURI: Specifies the location, relative to the provider class path, where the MBean XML descriptor file is located.

type: Specifies the type to use for registering this MBean. The type must match the type that is declared in the MBean descriptor file.

Java Management Extensions connector properties

A Java Management Extensions (JMX) connector can either be a Remote Method Invocation (RMI) connector or a Simple Object Access Protocol (SOAP) connector.

Depending on the property, you can specify or set a property in the administrative console, the wsadmin tool, Application Server commands, scripts run from a command line interface, or a custom Java administrative client program that you write. You can also set SOAP connector properties in the `soap.client.props` file.

For specific information on how to code the JMX connector properties for the wsadmin tool, the Application Server commands, or scripts, see the particular tool or command. For specific information on how to code the JMX connector properties for a custom Java administrative client program, see the ″Java API documentation for Application Server″ topic in the Information Center.

For the administrative console, this article specifies the coding of the particular setting or property. Coding of properties in the `soap.client.props` file that are specific to JMX connectors is specified. These properties begin with com.ibm.SOAP. Other properties in the `soap.client.props` file that contain information that can be set elsewhere in the Application Server are not documented here. The coding for the com.ibm.ssl.contextProvider property, which can be set only in the `soap.client.props` file, is specified.

Each profile has a property file at *installation root*/profiles/*profile name*/properties/soap.client.props. These property files allow you to set different properties, including security and timeout properties. These properties are the default for all administrative connections that use the SOAP JMX connector between processes executing in a particular profile. For instance, the wsadmin program executing under a particular profile uses the property values from that file for the SOAP connector behavior unless the properties are overridden by some other programmatic means.

To view the JMX connector custom properties administrative console panel that goes with this article, click one of the following paths:

- **Servers -> Application servers ->***server name* **-> Server Infrastructure -> Administration -> Administration Services -> Additional properties -> JMX Connectors->***connector type* **-> Additional Properties -> Custom properties**
- **System administration -> Deployment manager ->Additional Properties -> Administration Services -> Additional Properties -> JMX Connectors->***connector type***-> Additional Properties -> Custom properties**
- **System administration -> Node agents ->***node agent name* **-> Additional Properties -> Administration Services -> Additional Properties -> JMX Connectors->***connector type***-> Additional Properties -> Custom properties**

SOAP connector properties

This section discusses JMX connector properties that pertain to SOAP connectors.

SOAP Request timeout

Specifies the SOAP client request timeout. The value that you choose depends on a number of factors such as the size and the number of the applications that are installed on the server, the speed of your machine, and the level of usage of your machine.

The program default value for the request timeout is 600 seconds. However, other components that connect to the SOAP client can override the default. Components that use the `soap.client.props` file have a default value of 180 seconds.

You can set the property by using one of the following options:

- Scripts run from a command line interface.
- The `soap.client.props` file.

Property	com.ibm.SOAP.requestTimeout
Data type	Integer
Range in seconds	0 to n If the property is zero (0), the request never times out.
Default	180

- The administrative console. Specify the property and the value as a name-value pair on the JMX connector custom properties panel of the administrative console.

Property	requestTimeout
Data type	Integer
Range in seconds	0 to n If the property is zero (0), the request never times out.
Default	600

- A Java administrative client. The property is AdminClient.CONNECTOR_SOAP_REQUEST_TIMEOUT.

Configuration URL

Specifies the Universal Resource Locator (URL) of the `soap.client.props` file. Specify the configuration URL property if you want a program to read SOAP properties from this file. You can set the property by using one of the following options:

- Scripts run from a command line interface. Scripts can pass the Configuration URL property to the Application Server on the com.ibm.SOAP.ConfigURL system property.
- The administrative console. Specify the property and the value as a name-value pair on the JMX connector custom properties panel of the administrative console.

Property	ConfigURL
Data type	String
Valid Value	http://*Path*/soap.client.props
Default	None

- A Java administrative client. The property is AdminClient.CONNECTOR_SOAP_CONFIG.

Security context provider

Specifies the Secure Sockets Layer (SSL) implementation to use between the Application Server and the SOAP client. You can specify either IBM Java Secure Sockets Extension (IBMJSSE) or IBM Java Secure Sockets Extension that has undergone Federal Information Processing Standards certification (IBMJSSEFIPS).

You can set the property by using the `soap.client.props` file.

Property	com.ibm.ssl.contextProvider
Data type	String

Valid Values	`IBMJSSE`
	`IBMJSSEFIPS`
	`IBMJSSE2`

Default	IBMJSSE2

Secure Sockets Layer (SSL) security

Use this property to enable SSL security between Application Server and the SOAP client. You can set the property by using one of the following options:

- Scripts run from a command line interface.
- The `soap.client.props` file.

Property	com.ibm.SOAP.securityEnabled
Data type	Boolean
Default	False

- The administrative console. Specify the property and the value as a name-value pair on the JMX connector custom properties panel of the administrative console.

Property	securityEnabled
Data type	Boolean
Default	False

- A Java administrative client. The property is AdminClient.CONNECTOR_SECURITY_ENABLED.

SOAP and RMI connector properties

This section discusses JMX connector properties that pertain to both SOAP connectors and RMI connectors.

Connector type

Specify a connector type of SOAP or RMI, depending on whether Application Server connects to a SOAP server or an RMI server. You can set the property by using one of the following options:

- The wsadmin tool.
- Scripts run from a command line interface.
- The administrative console. Specify the property and the value as a name-value pair on the JMX connector custom properties panel of the administrative console.

Property	Type
Data type	String
Valid values	`SOAPConnector` `RMIConnector`
Default	SOAP

- A Java administrative client. The property is AdminClient.CONNECTOR_TYPE. Specify by using the AdminClient.CONNECTOR_TYPE_RMI or the AdminClient.CONNECTOR_TYPE_SOAP constants.

Host

Use the host property to specify the host name or the IP address of the server to which Application Server connects. The server can be a SOAP server or an RMI server. You can set the property by using one of the following options:

- The wsadmin tool.
- Scripts run from a command line interface.
- The administrative console. Specify the property and the value as a name-value pair on the JMX connector custom properties panel of the administrative console.

Property	host
Data type	String
Valid values	Host name or IP address
Default	None

- A Java administrative client. The property is AdminClient.CONNECTOR_HOST.

Port

Use the port property to specify the port number of the server to which Application Server connects. The server can be a SOAP server or an RMI server. You can set the property by using one of the following options:

- The wsadmin tool.
- Scripts run from a command line interface.

- The administrative console. Specify the property and the value as a name-value pair on the JMX connector custom properties panel of the administrative console.

Property	port
Data type	Integer
Valid value	Port number
Default	None

- A Java administrative client. The property is AdminClient.CONNECTOR_PORT.

User name

Specifies the user name that Application Server uses to access the SOAP server or the RMI server. You can set the property by using one of the following options:

- The wsadmin tool.
- Scripts run from a command line interface.
- The `soap.client.props` file.

Property	com.ibm.SOAP.loginUserid
Data type	String
Valid value	The value must match the global SSL settings for SOAP or RMI.
Default	None

- The administrative console. Specify the property and the value as a name-value pair on the JMX connector custom properties panel of the administrative console.

Property	username
Data type	String
Valid value	The value must match the global SSL settings for SOAP or RMI.
Default	None

- A Java administrative client. The property is AdminClient.USERNAME.

Password

Specifies the password that Application Server uses to access the SOAP server or the RMI server. You can set the property by using one of the following options:

- The wsadmin tool.
- Scripts run from a command line interface.
- The `soap.client.props` file.

Property	com.ibm.SOAP.loginPassword
Data type	String
Valid values	The value must match the global SSL settings for SOAP or RMI.
Default	None

- The administrative console. Specify the property and the value as a name-value pair on the JMX connector custom properties panel of the administrative console.

Property	password
Data type	String

Valid values	The value must match the global SSL settings for SOAP or RMI.
Default	None

- A Java administrative client. The property is AdminClient.PASSWORD.

Java Management Extensions connectors

Use this page to view and change the configuration for Java Management Extensions (JMX) connectors.

To view this administrative console page, click **Servers > Application Servers >***server_name* **> Administration > Administration Services > JMX Connectors**

Java Management Extensions (JMX) connectors communicate with WebSphere Application Server when you invoke a scripting process. There is no default for the type and parameters of a connector. The `wsadmin.properties` file specifies the Simple Object Access Protocol (SOAP) connector and an appropriate port number. You can also use the Remote Method Invocation (RMI) connector.

Use one of the following methods to select the connector type and attributes:
- Specify properties in a properties file.
- Indicate options on the command line.

Type:

Specifies the type of the JMX connector.

Data type	Enum
Default	SOAPConnector
Range	**SOAPConnector** For JMX connections using Simple Object Access Protocol (SOAP). **RMIConnector** For JMX connections using Remote Method Invocation (RMI).

JMX connector settings:

Use this page to view the configuration for a Java Management Extensions (JMX) connector.

To view this administrative console page, click **Servers > Application Servers >***server_name* **> Administration Services > JMX Connectors >***connector_type*

Type:

Specifies the type of the JMX connector.

Data type	Enum
Default	SOAPConnector
Range	**SOAPConnector** For JMX connections using Simple Object Access Protocol (SOAP). **RMIConnector** For JMX connections using Remote Method Invocation (RMI).

Repository service settings

Use this page to view and change the configuration for an administrative service repository.

To view this administrative console page, click **Servers > Application Servers >***server_name* **> Administration Services > Repository Service**.

Audit Enabled:

Specifies whether to audit repository updates in the log file. The default is to audit repository updates.

Data type	Boolean
Default	true

Administrative agents: Resources for learning

Use the following links to find relevant supplemental information about WebSphere Application Server administrative agents and distributed administration. 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:

Administration

- IBM WebSphere Application Server Redbooks at `http://publib-b.boulder.ibm.com/Redbooks.nsf/portals/WebSphere`.

  This site contains a listing of all WebSphere Application Server Redbooks.
- IBM developerWorks WebSphere at `http://www.software.ibm.com/wsdd/`.

  This site is the home of technical information for developers working with WebSphere products. You can download WebSphere software, take a fast path to developerWorks zones, such as VisualAge Java or WebSphere Application Server, learn about WebSphere products through a newcomers page, tutorials, technology previews, training, and Redbooks, get answers to questions about WebSphere products, and join the WebSphere community, where you can keep up with the latest developments and technical papers.
- WebSphere Application Server Support page at `http://www.ibm.com/software/webservers/appserv/support.html`.

  Take advantage of the Web-based Support and Service resources from IBM to quickly find answers to your technical questions. You can easily access this extensive Web-based support through the IBM Software Support portal and search by product category, or by product name. For example, if you are experiencing problems specific to WebSphere Application Server, click **WebSphere Application Server** in the product list. The WebSphere Application Server Support page appears.

Configuring the environment

To assist in handling requests among Web applications, Web containers, and application servers, you can configure settings for virtual hosts, variables and shared libraries.

1. Configure virtual hosts.
2. Configure variables.
3. If your deployed applications use shared library files, define the shared library files needed.

   See “Managing shared libraries” on page 97.

Virtual hosts

A virtual host is a configuration 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.

Each virtual host has a logical name and a list of one or more DNS aliases by which it is known. A DNS alias is the TCP/IP hostname and port number that is used to request the servlet, for example yourHostName:80. When no port number is specified, 80 is assumed.

When a servlet request is made, the server name and port number entered into the browser are compared to a list of all known aliases in an effort to locate the correct virtual host and serve the servlet. If no match is found, an error is returned to the browser.

An application server provides a default virtual host with some common aliases, such as the machine's 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:80 in the request http://localhost:80/myServlet.

A virtual host is not associated with a particular machine. It is a configuration, rather than a ″live object,″ explaining why you can create it, but cannot start or stop it. For many users, creating virtual hosts is unnecessary because the default_host is provided.

Adding a localhost to the virtual hosts adds the host name and IP address of the localhost machine to the alias table. This allows a remote user to access the administrative console.

Why you would use virtual hosting

Virtual hosts let you manage a single application server on a single machine as if the application server were multiple application servers each on their own host machine. Resources associated with one virtual host cannot share data with resources associated with another virtual host. This is true even though the virtual hosts share the same application server on the same physical machine.

Virtual hosts allow the administrator to isolate and independently manage multiple sets of resources on the same physical machine.

Suppose an Internet service provider (ISP) has two customers with Internet sites hosted on the same machine. The ISP keeps the two sites isolated from one another, despite their sharing a machine, by using virtual hosts. The ISP associates the resources of the first company with VirtualHost1 and the resources of the second company with VirtualHost2. Both virtual hosts map to the same application server.

Further suppose that both company sites offer the same servlet. Each site has its own instance of the servlet, and is unaware of the same servlet on the other site. If the company whose site is organized on VirtualHost2 is past due in paying its account with the ISP, the ISP can refuse all servlet requests that are routed to VirtualHost2. Even though the same servlet is available on VirtualHost1, the requests directed at VirtualHost2 do not go to the other virtual host.

The servlets on one virtual host do not share their context with the servlets on the other virtual host. Requests for the servlet on VirtualHost1 can continue as usual. This is true even though VirtualHost2 is refusing to fill requests for the servlet with the same name.

You associate a servlet or other application with a virtual host instead of the actual DNS address.

The default virtual host (default_host)

The product provides a default virtual host (named default_host).

The virtual host configuration uses wildcard entries with the ports for its virtual host entries.

- The default alias is *:80, using an internal port that is not secure.
- Aliases of the form *:9080 use the secure internal port.
- Aliases of the form *:9443 use the external port that is not secure.
- Aliases of the form *:443 use the secure external port.

Unless you specifically want to isolate resources from one another on the same physical machine, you probably do not need any virtual hosts in addition to the default host.

How requests map to virtual host aliases

Virtual hosts let you manage a single application server on a single machine as if the application server were multiple application servers that are each on their own host machine. Resources associated with one virtual host cannot share data with resources associated with another virtual host, even though the virtual hosts share the same application server on the same physical machine.

When you request a resource, WebSphere Application Server tries to map the request to an alias of a defined virtual host.

Mappings are both case sensitive and insensitive. For example, the portion ″http://host:port/″ is case insensitive, but the URL that follows is case sensitive. The match must be alphanumerically exact. Also, different port numbers are treated as different aliases.

For example, the request `http://www.myhost.com/myservlet` maps successfully to `http://WWW.MYHOST.COM/myservlet` but not to `http://WWW.MYHOST.COM/MYSERVLET or Www.Myhost.Com/Myservlet`. In the latter two cases, these mappings fail due to case sensitivity. The request `http://www.myhost.com/myservlet` does not map successfully to `http://myhost/myservlet` or to `http://myhost:9876/myservlet`. These mappings fail because they are not alphanumerically correct.

You can use wildcard entries for aliases by port and specify that all valid host name and address combinations on a particular port map to a particular virtual host.

If you request a resource using an alias that cannot be mapped to an alias of a defined virtual host, you receive a 404 error in the browser that was used to issue the request. A message states that the virtual host could not be found.

Two sets of associations occur for virtual hosts. Application deployment associates an application with a virtual host. Virtual host definitions associate the network address of the machine and the HTTP transport or Web server port assignment of the application server with the virtual host. Looking at the flow from the Web client request for the snoop servlet, for example, the following actions occur:

1. The Web client asks for the snoop servlet: at Web address http://www.some_host.some_company.com:9080/snoop
2. The some_host machine has the 9080 port assigned to the stand-alone WebSphere Application Server, *server1*.
3. The server1 Application Server looks at the virtual host assignments to determine the virtual host that is assigned to the alias some_host.some_company.com:9080.
4. The application server finds that no explicit alias for that DNS string exists. However, a wild card assignment for host name * at port 9080 does exist. This is a match. The virtual host that defines the match is default_host.
5. The application server looks at the applications deployed on the default_host and finds the snoop servlet.
6. The application server serves the application to the Web client and the requester is able to use the snoop servlet.

You can have any number of aliases for a virtual host. You can even have overlapping aliases, such as:

Virtual host	Alias	Port
default_host	*	9080
	localhost	9080
	my_machine	9080
	my_machine.my_company.com	9080
	localhost	80

The Application Server looks for a match using the explicit address specified on the Web client address. However, it might resolve the match to any other alias that matches the pattern before matching the explicit address. Simply defining an alias first in the list of aliases does not guarantee the search order when WebSphere Application Server is looking for a matching alias.

A problem can occur if you use the same alias for two different virtual hosts. For example, assume that you installed the default application and the snoop servlet on the default_host. You also have another virtual host called the admin_host. However, you have not installed the default application or the snoop servlet on the admin_host.

Assume that you define overlapping aliases for both virtual hosts because you accidentally defined port 9080 for the admin_host instead of port 9060:

Virtual host	Alias	Port
default_host	*	9080
	localhost	9080
admin_host	*	9060
	my_machine.com	9080

Assume that a Web client request comes in for http://my_machine.com:9080/snoop.

If the application server matches the request against *:9080, the application is served from the default_host. If the application server matches the request to my.machine.com:9080, the application cannot be found. A 404 error occurs in the browser that issues the request. A message states that the virtual host could not be found.

This problem is the result of not finding the requested application in the first virtual host that has a matching alias. The correct way to code aliases is for the alias name on an incoming request to match only one virtual host in all of your virtual host definitions. If the URL can match more than one virtual host, you can see the problem just described.

Configuring virtual hosts

Virtual hosts enable you to isolate and independently manage multiple sets of resources on the same physical machine.

1. Create a virtual host using the “Virtual host collection” on page 90 of the administrative console. Click **Environment > Virtual Hosts** from the navigation tree of the console. Click **New**. On the “Virtual host settings” on page 91 page that displays, specify an administrative name for the virtual host. When you create a virtual host, a default set of 90 MIME entries is created for the virtual host.
2. There must be a virtual host alias corresponding to each port used by an HTTP transport. There is one HTTP transport in each Web container, usually assigned to the virtual host named default_host. You can change the default assignment to any valid virtual host.

You must create a virtual host for each HTTP port in the following cases:

- You are using the internal HTTP transport with a port other than the default of 9080. You must define the port that you are using.
- You are using the default port 9080, but the port is no longer defined. You must define port 9080.
- You have created multiple Application Servers (either stand-alone servers or cluster members) that use the same virtual host. Because each server must be listening on a different HTTP transport port, you must define a virtual host alias for the transport port of each server.

If you define new virtual host aliases, identify the port values that the aliases use on the “HTTP transport collection” on page 130 page.

3. If necessary, create a virtual host alias for each HTTP transport port.

   From the “Virtual host collection” page, click your virtual host. On the “Virtual host settings” on page 91 page for the virtual host, click **Host aliases**. To define a virtual host alias on the “Host alias collection” on page 91 page, click **New**. On the “Host alias settings” on page 92 page for the virtual host alias, specify a host name and a port. Configure the virtual host to contain an alias for the port number. For example, specify an alias of `*:9082` if 9082 is the port number in use by the transport.

4. When you enter the URL for the application into a Web browser, include the port number in the URL.

   For example, if 9082 is the port number, specify a URL such as `http://localhost:9082/wlm/SimpleServlet`

5. If MIME entries are not specified at the Web module level, define MIME object types and their file name extensions. For each needed MIME entry on the “MIME type collection” on page 92 page, click **New**. On the “MIME type settings” on page 93 page, specify a MIME type and extension.

6. After you configure a virtual host alias or change a configuration, you must regenerate the Web server plug-in configuration and restart WebSphere Application Server.

Virtual host collection

Use this page to create and manage configurations that each let a single host machine resemble multiple host machines. Such configurations are known as *virtual hosts.*

To view this administrative console page, click **Environment > Virtual Hosts**.

Each virtual host has a logical name (which you define on this panel) and is known by its list of one or more domain name system (DNS) aliases. A DNS alias is the TCP/IP host name and port number used to request the servlet, for example yourHostName:80. (Port 80 is the default.)

You define one or more alias associations by clicking an existing virtual host or by adding a new virtual host.

When a servlet request is made, the server name and port number entered into the browser are compared to a list of all known aliases in an effort to locate the correct virtual host to serve the servlet. No match returns an error to the browser.

An application server profile provides a default virtual host with some common aliases, such as the internet protocol (IP) address, the DNS short host name, and the DNS fully qualified host name. The alias comprises the first part of the path for accessing a resource such as a servlet.

For example, the alias is localhost:80 in the request http://localhost:80/myServlet.

A virtual host is not associated with a particular profile or node (machine), but is associated with a particular server instead. It is a configuration, rather than a ″live object.″ You can create a virtual host, but you cannot start or stop it.

For many users, creating virtual hosts is unnecessary because the default_host that is provided is sufficient.

Adding the host name and IP address of the localhost machine to the alias table lets a remote user access the administrative console.

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.

Name:

Specifies a logical name for configuring Web applications to a particular host name. The default virtual host is suitable for most simple configurations.

Virtual hosts enable you to isolate, and independently manage, multiple sets of resources on the same physical machine. Determine whether you need a virtual host alias for each port associated with an HTTP transport channel or an HTTP transport. There must be a virtual host alias corresponding to each port used by an HTTP transport channel or an HTTP transport. There is one HTTP transport channel or HTTP transport associated with each Web container, and there is one Web container in each application server.

When you create a virtual host, a default set of 90 MIME entries is created for the virtual host.

You must create a virtual host for each HTTP port in the following cases:
- You use the internal HTTP transport with a port other than the default value of 9080, or for some reason the virtual host does not contain the usual entry for port 9080.
- You create multiple application servers (stand-alone servers, managed servers, or cluster members) that are using the same virtual host. Because each server must be listening on a different HTTP port, you need a virtual host alias for the HTTP port of each server.

Virtual host settings:

Use this page to configure a virtual host instance.

To view this administrative console page, click **Environment** > **Virtual Hosts** >*virtual_host_name*.

Name:

Specifies a logical name for configuring Web applications to a particular host name. The default virtual host is suitable for most simple configurations.

Data type	String
Default	default_host

Host alias collection:

Use this page to manage host name aliases defined for a virtual host. An alias is the DNS host name and port number that a client uses to form the URL request for a Web application resource.

To view this administrative console page, click **Environment** > **Virtual Hosts** >*virtual_host_name* > **Host Aliases**.

Host Name:

Specifies the IP address, DNS host name with domain name suffix, or just the DNS host name, used by a client to request a Web application resource (such as a servlet, JavaServer Pages (JSP) file, or HTML page). For example, the host alias name is `myhost` in a DNS name of `myhost:8080`.

The product provides a default virtual host (named default_host). The virtual host configuration uses the wildcard character * (asterisk) along with the port number for its virtual host entries. Unless you specifically want to isolate resources from one another on the same node (physical machine), you probably do not need any virtual hosts in addition to the default host.

Port:

Specifies the port for which the Web server has been configured to accept client requests. For example, the port assignment is *8080* in a DNS name of `myhost:8080`. A URL refers to this DNS as: http://*myhost:8080*/servlet/snoop.

Host alias settings:

Use this page to view and configure a host alias.

To view this administrative console page, click **Environment > Virtual Hosts >***virtual_host_name* **> Host Aliases >***host_alias_name*.

Host name:

Specifies the IP address, domain name system (DNS) host name with domain name suffix, or the DNS host name that clients use to request a Web application resource, such as a servlet, JSP file, or HTML page.

For example, when the DNS name is `myhost`, the host alias is `myhost:8080`, where *8080* is the port. A URL request can refer to the snoop servlet on the host alias as: http://*myhost:8080*/servlet/snoop.

When there is no port number specified for a host alias, the default port is 80. For existing virtual hosts, the default host name and port reflect the values specified at product installation or configuration. For new virtual hosts, the default can be `*` to allow any value or no specification.

Data type	String
Default	* You can also use the IP address or the long or short DNS name.

Port:

Specifies the port where the Web server accepts client requests. Specify a port value in conjunction with the host name.

The default reflects the value specified at product setup. The default might be `80`, `81`, `9060` or a similar value.

Data type	Integer
Default	9060

MIME type collection:

Use this page to view and configure multi-purpose internet mail extensions (MIME) object types and their file name extensions.

The list shows a collection of MIME type extension mappings defined for the virtual host. Virtual host MIME entries apply when you do not specify MIME entries at the Web module level.

To view a list of current virtual host Mime types in the administrative console, click **Environment > Virtual Hosts >***virtual_host_name* **> MIME Types**.

MIME Type:

Specifies a MIME type, which can be application, audio, image, text, video, www, or x-world. An example value for MIME type is `text/html`.

Extensions:

Specifies file extensions of files that map the MIME type. Do not specify the period before the extension. Example extensions for a `text/html` MIME type are `htm` and `html`.

MIME type settings:

Use this page to configure a multi-purpose internet mail extensions (MIME) object type.

To view this administrative console page, click **Environment > Virtual Hosts >***virtual_host_name* **> MIME Types >***MIME_type*.

MIME Type:

Specifies a MIME type, which can be application, audio, image, text, video, www, or x-world. An example value for MIME type is `text/html`.

An example value for MIME type is `text/html`. A default value appears only if you are viewing the configuration for an existing instance.

Data type	String

Extensions:

Specifies file extensions of files that map the MIME type. Do not specify the period before the extension. Example extensions for a `text/html` MIME type are `htm` and `html`.

File extensions for a `text/html` MIME type are `.htm` and `.html`. A default value appears only if you are viewing the configuration for an existing MIME type.

Data type	String

Variables

A variable is a configuration property that can be used to provide a parameter for some values in the system. A variable has a name and a value.

WebSphere variables are used for:

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

Each variable has a scope. A scope is the range of locations in the WebSphere Application Server network where the variable is applicable.

- A variable with a node-level scope is available only on the node and the servers on that node. If a node-level variable has the same name as a cell-wide variable, the node-level variable value takes precedence.

- A server variable is available only on the one server process. A server variable takes precedence over a variable with the same name that is defined at a higher level.

You can use variables in configuration values such as file system path settings. Use the following syntax to refer to a variable:

```
${variable_name}
```

The value of a variable can contain a reference to another variable. The value of the variable is computed by substituting the value of the referenced variable recursively.

Variables are useful when concatenating two path variables when the specification does not accept the *AND* operator. For example, suppose that the following variables exist:

Variable name	Variable value
ROOT_DIR	`/`
HOME_DIR	`${ROOT_DIR}home`
USER_DIR	`${HOME_DIR}/myuserdir`

The variable reference `${USER_DIR}` resolves to the value `/home/myuserdir`.

Configuring WebSphere variables

This topic describes how to create a WebSphere Application Server variable.

You can define a WebSphere Application Server variable to provide a parameter for some values in the system. After you define the name and value for a variable, the value is used in place of the variable name. Variables most often specify file paths. However, some system components also support the use of variables.

WebSphere variables are used for:

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

The scope of a variable can be cell-wide, node-wide, or applicable to only one server process.

Define variables on the **Environment > WebSphere Variables** console page.

Define the scope to apply a variable node-wide or to only one server process. A variable resolves to its new value when used in a component that supports the use of variables.

1. Click **Environment > WebSphere Variables** in the administrative console to define a new variable.
2. Specify the scope of the variable. Declare the new variable for the **Node** or **Server** and click **Apply**.

   The variable exists at the level you specify. Define a variable at multiple levels to use multiple values. The more granular definition overrides the higher level setting.

   For instance, if you specify the same variable on a node and a server, the server setting overrides the node setting.

   Scoping variables is particularly important when testing data source objects. Variable scoping can cause a data source to fail the test connection, but to succeed at run time, or to pass the test connection, but fail at run time.

   See the *Developing and deploying applications* PDF for more information.
3. Click **New** on the WebSphere Variables page.
4. Specify a name, a value, and a description on the Variable page. Click **OK**.

5. Verify that the variable is displayed in the list of variables. The administrative console does not pick up typing errors. The variable is ignored if it is referred to incorrectly.
6. Save your configuration.
7. Stop the server and start the server again to put the variable configuration into effect.

WebSphere variables collection

Use this page to view and change a list of substitution variables with their values and scope.

To view this administrative console page, click **Environment > WebSphere Variables**.

For information on a variable, click the variable and read the value in the **Description** field.

Name:

Specifies the symbolic name for a WebSphere Application Server variable. For example, a variable name might represent a physical path or URL root used by WebSphere Application Server.

Value:

Specifies the value that the symbolic name represents. For example, the value might be an absolute path value for a file or URL root.

Scope:

Specifies the level at which a WebSphere Application Server variable is visible on the administrative console panel.

A resource can be visible in the administrative console collection table at the node or server scope.

Variable settings:

Use this page to define the name and value of a WebSphere Application Server substitution variable.

To view this administrative console page, click **Environment > WebSphere Variables >***WebSphere_variable_name*.

Name:

Specifies the symbolic name for a WebSphere Application Server variable. For example, a variable name might represent a physical path or URL root that is used by WebSphere Application Server.

WebSphere variables are used for:

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

WebSphere Application Server substitutes the symbolic name wherever its value displays in the system.

For example, ″JAVA_HOME″ is the symbolic name representing the file system path to the installation directory for the Java Virtual Machine (JVM). For example, the value is `/opt/IBM/WebSphere/AppServer/java` for the WebSphere Application Server product on a Linux machine.

You can create new variables for use in WebSphere Application Server components that support the use of variables.

Data type	String

Value:

Specifies the value that the symbolic name represents. For example, the value might be an absolute path value for a file or URL root.

For example, `/opt/IBM/WebSphere/AppServer/java` is the value on a Linux machine for a variable named JAVA_HOME.

Data type	String

Description:

Documents the purpose of a variable.

Data type	String

IBM Toolbox for Java JDBC driver

WebSphere Application Server supports the **IBM Toolbox for Java** JDBC driver. The IBM Toolbox for Java JDBC driver is included with the IBM Toolbox for Java product.

IBM Toolbox for Java is a library of Java classes that are optimized for accessing iSeries and AS/400 data and resources. You can use the IBM Toolbox for Java JDBC driver to access local or remote **DB2 UDB for iSeries 400** databases from server-side and client Java applications that run on any platform that supports Java.

IBM Toolbox for Java is available in these versions:

IBM Toolbox for Java licensed program

The licensed program is available with every OS/400 release, starting with Version 4 Release 2 (V4R2). You can install the licensed program on your iSeries 400 system, and then either copy the IBM Toolbox for Java JAR file (*jt400.jar*) to your system or update your system *classpath* to locate the server installation. Product documentation for IBM Toolbox for Java is available from the iSeries information center: `http://publib.boulder.ibm.com/iseries/v5r1/ic2924/index.htm` Locate the documentation by traversing the following path in the left-hand navigation window of the iSeries information center: **Programming** > **Java** > **IBM Toolbox for Java**.

JTOpen

JTOpen is the open source version of IBM Toolbox for Java, and is more frequently updated than the licensed program version. You can download JTOpen from `http://www-1.ibm.com/servers/eserver/iseries/toolbox/downloads.htm`. You can also download the *JTOpen Programming Guide*. The guide includes instructions for installing JTOpen and information about the JDBC driver.

The JDBC driver for both versions supports JDBC 2.0. For more information about IBM Toolbox for Java and JTOpen, see the product Web site at `http://www-1.ibm.com/servers/eserver/iseries/toolbox/index.html`.

Note: If you are using WebSphere Application Server on platforms other than iSeries, use the **JTOpen** version of the Toolbox JDBC driver.

Configure and use the jt400.jar file

1. Download the *jt400.jar* file from the **JTOpen** URL at `http://www-1.ibm.com/servers/eserver/iseries/toolbox/downloads.htm`. Place it in a directory on your workstation such as *C:\JDBC_Drivers\Toolbox*.
2. Open the administrative console.
3. Select **Environment**.
4. Select **Managed WebSphere Variables**.
5. Set the managed variable *OS400_TOOLBOX_JDBC_DRIVER_PATH* at the **Node** level.
6. Double click **OS400_TOOLBOX_JDBC_DRIVER_PATH**.
7. Set the value to the full directory path to the `jt400.jar` file downloaded in step one. Do not include *jt400.jar* in this value. For example,

   ```
   OS400_TOOLBOX_JDBC_DRIVER_PATH == "C:\JDBC_Drivers\Toolbox"
   ```

   When you choose a Toolbox driver from the list of possible resource providers the **Classpath** field looks like:

   ```
   Classpath == ${OS400_TOOLBOX_JDBC_DRIVER_PATH}/jt400.jar
   ```

Shared library files

Shared library files in WebSphere Application Server consist of a symbolic name, a Java class path, and a native path for loading Java Native Interface (JNI) libraries.

You can define a shared library at the cell, node, or server level. Defining a library at one of the three levels does not cause the library to be placed into the application server's class loader. You must associate the library to an application or server in order for the classes represented by the shared library to be loaded in either a server-wide or application-specific class loader.

A separate class loader is used for shared libraries that are associated with an application server. This class loader is the parent of the application class loader, and the WebSphere Application Server extensions class loader is its parent. Shared libraries that are associated with an application are loaded by the application class loader.

Managing shared libraries

Shared libraries are files used by multiple applications. Using the administrative console, you can define a shared library at the cell, node, or server level. You can then associate the library to an application or server to load the classes represented by the shared library in either a server-wide or application-specific class loader. Using an installed optional package, you can associate a shared library to an application by declaring the dependent library `.jar` file in the `MANIFEST.MF` file of the application. Refer to the Java 2 Platform, Enterprise Edition (J2EE) 1.4 specification, section 8.2 for an example.

If your deployed applications use shared library files, define shared libraries for the library files and associate the libraries with specific applications or with an application server. Associating a shared library file with a server associates the file with all applications on the server. Use the Shared Libraries page to define new shared library files to the system and remove them.

- Use the administrative console to define a shared library.
  1. Create a shared library for each library file that your applications need.
  2. Associate each shared library with an application or a server.
     - Associate a shared library with an application that uses the shared library file.
     - Associate a shared library with an application server so every application on the server can use the shared library file.
- Use an installed optional package to declare a shared library for an application.
- Remove a shared library.

1. Click **Environment > Shared Libraries** in the console navigation tree to access the Shared Libraries page.
2. Select the library to be removed.
3. Click **Delete**.

The list of shared libraries is refreshed. The library file no longer displays in the list.

Creating shared libraries

Shared libraries are files used by multiple applications.

The first step for making a library file available to multiple applications deployed on a server is to create a shared library for each library file that your applications need. When you create the shared libraries, set variables for the library files.

Use the Shared Libraries page to create and configure shared libraries.

1. Go to the Shared Libraries page. Click **Environment > Shared Libraries** in the console navigation tree.
2. Change the scope of the collection table to see what shared libraries are in a cell, node, or server.
   a. Select the cell, a node, or a server.
   b. Click **Apply**.
3. Click **New**.
4. Configure the shared library.
   a. On the settings page for a shared library, specify the name, class path, and any other variables for the library file that are needed.
   b. Click **Apply**.
5. Repeat steps 1 through 4 until you define a shared library instance for each library file that your applications need.

Using the administrative console, associate your shared libraries with specific applications or with the class loader of an application server. Associating a shared library file with a server class loader associates the file with all applications on the server.

Alternatively, you can use an installed optional package to associate your shared libraries with an application.

Shared library collection

Use this page to define a list of shared library files that deployed applications can use.

To view this administrative console page, click **Environment > Shared Libraries**.

By default, a shared library is accessible to applications deployed (or installed) on the same node as the shared library file. Use the **Scope** field to change the scope to a different node or to a specific server.

Name:

Specifies a name for the shared library.

Description:

Describes the shared library file.

Shared library settings:

Use this page to make a library file available to deployed applications.

To view this administrative console page, click **Environment > Shared Libraries >***shared_library_name*.

Name:

Specifies a name for the shared library.

Data type	String

Description:

Describes the shared library file.

Data type	String

Classpath:

Specifies the class path used to locate the JAR files for the shared library support.

Data type	String
Units	Class path

Native Library Path:

Specifies the class path for locating platform-specific library files for shared library support; for example, .dll, .so, or *SRVPGM objects.

Data type	String
Units	Class path

Associating shared libraries with applications

You can associate a shared library with an application. Classes represented by the shared library are then loaded in the application's class loader, making the classes available to the application.

This article assumes that you have defined a shared library at the cell, node, or server level. The shared library represents a library file used by multiple deployed applications.

This article also assumes that you want to use the administrative console, and not an installed optional package, to associate a shared library with an application.

To associate a shared library with an application, create and configure a library reference using the administrative console. A library reference specifies the name of the shared library file.

If you associate a shared library with an application, do not associate the same shared library with a server class loader.

1. Click **Applications > Enterprise Applications >***application_name* **> Libraries** in the console navigation tree to access the Library Ref page.
2. Click **Add**.
3. On the settings page for a library reference, specify variables for the library reference as needed. The variables identify the shared library file that your application uses.
4. Click **Apply**.

The name of the library reference is shown in the list on the Library Ref page.

Repeat steps 2 through 4 until you define a library reference instance for each shared library that your application requires.

Associating shared libraries with servers

You can associate shared libraries with the class loader of a server. Classes represented by the shared library are then loaded in a server-wide class loader, making the classes available to all applications deployed on the server.

This article assumes that you have defined a shared library at the cell, node, or server level. The shared library represents a library file used by multiple deployed applications.

To associate a shared library with the class loader of a server, create and configure a library reference using the administrative console. A library reference specifies the name of the shared library file.

If you associate a shared library with a server class loader, do not associate the same shared library with an application.

1. Configure class loaders for applications deployed on the server.
   a. Click **Servers > Application Servers > *server_name*** to access the settings page for the application server.
   b. Set values for the application **Class loader policy** and **Class loading mode** of the server. For information on these settings, see "Application server settings" on page 115 and class loaders.
2. Create a library reference for each shared library file that your application needs.
   a. Go to the settings page for a class loader.
   b. Click **Libraries** to access the Library Ref page.
   c. Click **Add**.
   d. On the settings page for a library reference, specify variables for the library reference as needed. The variables identify the shared library file that your application uses.
   e. Click **Apply**. The name of the library reference is shown in the list on the Library Ref page.

   Repeat the previous steps until you define a library reference for each shared library that your application needs.

Installed optional packages

Installed optional packages enable applications to use the classes in Java archive (`.jar`) files without having to include them explicitly in a class path. An installed optional package is a `.jar` file containing specialized tags in its manifest file that enable the application server to identify it. An installed optional package declares one or more shared library `.jar` files in the manifest file of an application. When the application is installed on a server, the classes represented by the shared libraries are loaded in the class loader of the application, making the classes available to the application.

When a Java 2 Platform, Enterprise Edition (J2EE) application is installed on a server, dependency information is specified in its manifest file. WebSphere Application Server reads the dependency information of the application (`.ear` file) to automatically associate the application with an installed optional package `.jar` file. WebSphere Application Server adds the `.jar` files in associated optional packages to the application class path. Classes in the installed optional packages are then available to application classes.

Installed optional packages used by WebSphere Application Server are described in section 8.2 of the J2EE specification, Version 1.4 at `http://java.sun.com/j2ee/j2ee-1_4-fr-spec.pdf`.

WebSphere Application Server supports using the manifest file (`manifest.mf`) in shared library `.jar` files and application `.ear` files. WebSphere Application Server does not support the Java 2 Platform Standard Edition (J2SE) Installed Optional Package semantics used in the J2SE specification (http://java.sun.com/j2se/1.3/docs/guide/extensions/spec.html), which primarily serve the applet environment. WebSphere Application Server ignores applet-specific tags within manifest files.

Sample manifest.mf file

A sample manifest file follows for an application app1.ear that refers to a single shared library file util.jar:

```
app1.ear:
    META-INF/application.xml
    ejb1.jar:
         META-INF/MANIFEST.MF:
             Extension-List: util
             util-Extension-Name: com/example/util
             util-Specification-Version: 1.4
         META-INF/ejb-jar.xml

util.jar:
    META-INF/MANIFEST.MF:
        Extension-Name: com/example/util
        Specification-Title: example.com's util package
        Specification-Version: 1.4
        Specification-Vendor: example.com
        Implementation-Version: build96
```

The syntax of a manifest entry depends on whether the entry applies to a member with a defining role (the shared library) or a member with a referencing role (a J2EE application or a module within a J2EE application).

Manifest entry tagging

Main tags used for manifest entries include the following:

Extension-List
A required tag with variable syntax. Within the context of the referencing role (application's manifest), this is a space delimited list that identifies and constructs unique Extension-Name, Extension-Specification tags for each element in the list. Within the context of the defining role (shared library), this tag is not valid.

Extension-Name
A required tag that provides a name and links the defining and referencing members. The syntax of the element within the referencing role is to prefix the element with the <ListElement> string. For each element in the Extension-List, there is a corresponding <ListElement>-Extension-Name tag. The defining string literal value for this tag (in the above sample com/example/util1) is used to match (in an equality test) the corresponding tags between the defining and referencing roles.

Specification-Version
A required tag that identifies the specification version and links the defining and referencing members.

Implementation-Version
An optional tag that identifies the implementation version and links the defining and referencing members.

Further information on these tags is in the .jar file specification at http://java.sun.com/j2se/1.4.2/docs/guide/jar/jar.html#Manifest%20Specification.

Using installed optional packages

You can associate one or more shared libraries with an application using an installed optional package that declares the shared libraries in the application's manifest file. Classes represented by the shared libraries are then loaded in the application's class loader, making the classes available to the application.

Read about installed optional packages in “Installed optional packages” on page 100 and in section 8.2 of the Java 2 Platform, Enterprise Edition (J2EE) specification, Version 1.4 at `http://java.sun.com/j2ee/j2ee-1_4-fr-spec.pdf`.

WebSphere Application Server does not support the Java 2 Platform Standard Edition (J2SE) Installed Optional Package semantics used in the J2SE specification (`http://java.sun.com/j2se/1.3/docs/guide/extensions/spec.html`), which primarily serve the applet environment. WebSphere Application Server ignores applet-specific tags within manifest files.

Installed optional packages expand the existing shared library capabilities of an application server. Prior to Version 6, an administrator was required to associate a shared library to an application or server. Installed optional packages enable an administrator to declare a dependency in an application’s manifest file to a shared library, with installed optional package elements listed in the manifest file, and automatically associate the application to the shared library. During application installation, the shared library `.jar` file is added to the class path of the application class loader.

If you use an installed optional package to associate a shared library with an application, do not associate the same shared library with an application class loader or a server class loader using the administrative console.

1. Assemble the library file, including the manifest information that identifies it as an extension. Two sample manifest files follow. The first sample manifest file has application `app1.ear` refer to a single shared library file `util.jar`:

   ```
   app1.ear:
       META-INF/application.xml
       ejb1.jar:
            META-INF/MANIFEST.MF:
                Extension-List: util
                util-Extension-Name: com/example/util
                util-Specification-Version: 1.4
            META-INF/ejb-jar.xml

   util.jar:
       META-INF/MANIFEST.MF:
           Extension-Name: com/example/util
           Specification-Title: example.com's util package
           Specification-Version: 1.4
           Specification-Vendor: example.com
           Implementation-Version: build96
   ```

   The second sample manifest file has application `app1.ear` refer to multiple shared library `.jar` files:

   ```
   app1.ear:
       META-INF/application.xml
       ejb1.jar:
            META-INF/MANIFEST.MF:
                Extension-List: util1 util2 util3
                Util1-Extension-Name: com/example/util1
                Util1-Specification-Version: 1.4
                Util2-Extension-Name: com/example/util2
                Util2-Specification-Version: 1.4
                Util3-Extension-Name: com/example/util3
                Util3-Specification-Version: 1.4
            META-INF/ejb-jar.xml

   util1.jar:
       META-INF/MANIFEST.MF:
           Extension-Name: com/example/util1
           Specification-Title: example.com's util package
           Specification-Version: 1.4
           Specification-Vendor: example.com
           Implementation-Version: build96
   ```

```
util2.jar:
    META-INF/MANIFEST.MF:
        Extension-Name: com/example/util2
        Specification-Title: example.com's util package
        Specification-Version: 1.4
        Specification-Vendor: example.com
        Implementation-Version: build96

util3.jar:
    META-INF/MANIFEST.MF:
        Extension-Name: com/example/util3
        Specification-Title: example.com's util package
        Specification-Version: 1.4
        Specification-Vendor: example.com
        Implementation-Version: build96
```

2. Create a shared library that represents the library file assembled in step 1. This installs the library file as a WebSphere Application Server shared library.
3. Assemble the application, declaring in the application manifest file dependencies to the library files named the manifest created for step 1.
4. Install the application on the server.

During application installation, the shared library `.jar` files are added to the class path of the application class loader.

Library reference collection

Use this page to view and manage library references that define how to use global libraries. For example, you can use this page to associate shared library files with a deployed application.

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

If no shared libraries are defined, after you click **Add** a message is displayed stating that you must define a shared library before you can create a library reference. A shared library is a container-wide library file that can be used by deployed applications. To define a shared library, click **Environment > Shared Libraries** and specify the scope of the container. Then, click **New** and specify a name and one or more paths for the shared library. After you define a shared library, return to this page, click **Add**, and create a library reference.

Library name:

Specifies a name for the library reference.

Library reference settings:

Use this page to define library references, which specify how to use global libraries.

To view this administrative console page, click **Applications > Enterprise Applications >***application_name* **> Libraries >***library_reference_name*. A shared library must be defined to view this page.

A shared library is a container-wide library file that can be used by deployed applications. To define a shared library, click **Environment > Shared Libraries** and specify the scope of the container. Then, click **New** and specify a name and one or more paths for the shared library.

Library name:

Specifies the name of the shared library to use for the library reference.

Data type	String

Environment: Resources for learning

Use the following links to find relevant supplemental information about configuring the WebSphere Application Server 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 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 instructions and examples

- WebSphere Application Server education at `http://www.ibm.com/software/websphere/technical`.

Administration

- Listing of all IBM WebSphere Application Server Redbooks at `http://publib-b.boulder.ibm.com/Redbooks.nsf/Portals/WebSphere`.

Working with server configuration files

Application server configuration documents 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.

1. Edit configuration files. The master repository is comprised of .xml configuration files. You can edit configuration files using the administrative console, scripting, wsadmin commands, programming, or by editing a configuration file directly.
2. Save changes made to configuration files. Using the console, you can save changes as follows:
   a. Click **Save** on the taskbar of the administrative console.
   b. On the Save page, click **Save**.
3. Handle temporary configuration files resulting from a session timing out.
4. Change the location of temporary configuration files.
5. Change the location of backed-up configuration files.
6. Change the location of temporary workspace files.
7. Back up and restore configurations.

Configuration documents

WebSphere Application Server stores configuration data for servers in several documents in a cascading hierarchy of directories. The configuration documents describe the available application servers, their configurations, and their contents. Most configuration documents have XML content.

Hierarchy of directories of documents

The cascading hierarchy of directories and the documents' structure support multinode replication to synchronize the activities of all servers in a cell. In a Network Deployment environment, changes made to configuration documents in the cell repository, are automatically replicated to the same configuration documents that are stored on nodes throughout the cell.

At the top of the hierarchy is the **cells** directory. It holds a subdirectory for each cell. The names of the cell subdirectories match the names of the cells. For example, a cell named *cell1* has its configuration documents in the subdirectory *cell1*.

On the Network Deployment node, the subdirectories under the cell contain the entire set of documents for every node and server throughout the cell. On other nodes, the set of documents is limited to what applies to that specific node. If a configuration document only applies to *node1*, then that document exists in the configuration on *node1* and in the Network Deployment configuration, but not on any other node in the cell.

Each cell subdirectory has the following files and subdirectories:

- The `cell.xml` file, which provides configuration data for the cell
- Files such as `security.xml`, `virtualhosts.xml`, `resources.xml`, and `variables.xml`, which provide configuration data that applies across every node in the cell
- The **clusters** subdirectory, which holds a subdirectory for each cluster defined in the cell. The names of the subdirectories under clusters match the names of the clusters.

  Each cluster subdirectory holds a cluster.xml file, which provides configuration data specifically for that cluster.
- The **nodes** subdirectory, which holds a subdirectory for each node in the cell. The names of the nodes subdirectories match the names of the nodes.

  Each node subdirectory holds files such as `variables.xml` and `resources.xml`, which provide configuration data that applies across the node. Note that these files have the same name as those in the containing cell's directory. The configurations specified in these node documents override the configurations specified in cell documents having the same name. For example, if a particular variable is in both cell- and node-level `variables.xml` files, all servers on the node use the variable definition in the node document and ignore the definition in the cell document.

  Each node subdirectory holds a subdirectory for each server defined on the node. The names of the subdirectories match the names of the servers. Each server subdirectory holds a `server.xml` file, which provides configuration data specific to that server. Server subdirectories might hold files such as `security.xml`, `resources.xml` and `variables.xml`, which provide configuration data that applies only to the server. The configurations specified in these server documents override the configurations specified in containing cell and node documents having the same name.
- The **applications** subdirectory, which holds a subdirectory for each application deployed in the cell. The names of the applications subdirectories match the names of the deployed applications.

  Each deployed application subdirectory holds a `deployment.xml` file that contains configuration data on the application deployment. Each subdirectory also holds a **META-INF** subdirectory that holds a J2EE application deployment descriptor file as well as IBM deployment extensions files and bindings files. Deployed application subdirectories also hold subdirectories for all .war and entity bean .jar files in the application. Binary files such as .jar files are also part of the configuration structure.

An example file structure is as follows:

```
cells
  cell1
     cell.xml resources.xml virtualhosts.xml variables.xml security.xml
     nodes
        nodeX
           node.xml variables.xml resources.xml serverindex.xml
           serverA
              server.xml variables.xml
           nodeAgent
              server.xml variables.xml
        nodeY
           node.xml variables.xml resources.xml serverindex.xml
     applications
        sampleApp1
           deployment.xml
           META-INF
```

```
          application.xml ibm-application-ext.xml ibm-application-bnd.xml
    sampleApp2
       deployment.xml
       META-INF
          application.xml ibm-application-ext.xml ibm-application-bnd.xml
```

Changing configuration documents

You can use one of the administrative tools (console, wsadmin, Java APIs) to modify configuration documents or edit them directly. It is preferable to use the administrative console because it validates changes made to configurations. ″“Configuration document descriptions”″ states whether you can edit a document using the administrative tools or must edit it directly.

Configuration document descriptions

Most configuration documents have XML content. The table below describes the documents and states whether you can edit them using an administrative tool or must edit them directly.

If possible, edit a configuration document using the administrative console because it validates any changes that you make to configurations. You can also use one of the other administrative tools (wsadmin or Java APIs) to modify configuration documents. Using the administrative console or wsadmin scripting to update configurations is less error prone and likely quicker and easier than other methods.

However, you cannot edit some files using the administrative tools. Configuration files that you must edit manually have an X in the **Manual editing required** column in the table below.

Document descriptions

(Locations split for publishing)

Configuration file	Locations	Purpose	Manual editing required
`admin-authz.xml`	`config/cells/` *`cell_name/`*	Define a role for administrative operation authorization.	X

`app.policy`	`config/cells/`
	`cell_name/`
	`nodes/`*`node_name/`*
		Define security permissions for application code.	X

`cell.xml`	`config/cells/` *`cell_name/`*	Identify a cell.	

`cluster.xml`	`config/cells/`
	`cell_name/`
	`clusters/`
	`cluster_name/`
		Identify a cluster and its members and weights.
		This file is only available with the Network Deployment product.	

`deployment.xml`	`config/cells/`
	`cell_name/`
	`applications/`
	`application_name/`
		Configure application deployment settings such as target servers and application-specific server configuration.	

`filter.policy`	`config/cells/` *`cell_name/`*	Specify security permissions to be filtered out of other policy files.	X
`integral-jms-authorizations.xml`	`config/cells/` *`cell_name/`*	Provide security configuration data for the integrated messaging system.	X

library.policy	config/cells/ *cell_name*/ nodes/*node_name*/	Define security permissions for shared library code.	X
multibroker.xml	config/cells/ *cell_name*/	Configure a data replication message broker.	
namestore.xml	config/cells/ *cell_name*/	Provide persistent name binding data.	X
naming-authz.xml	config/cells/ *cell_name*/	Define roles for a naming operation authorization.	X
node.xml	config/cells/ *cell_name*/ nodes/*node_name*/	Identify a node.	
pmirm.xml	config/cells/ *cell_name*/	Configure PMI request metrics.	X
resources.xml	config/cells/ *cell_name*/ config/cells/ *cell_name*/ nodes/*node_name*/ config/cells/ *cell_name*/ nodes/*node_name*/ servers/ *server_name*/	Define operating environment resources, including JDBC, JMS, JavaMail, URL, JCA resource providers and factories.	
security.xml	config/cells/ *cell_name*/	Configure security, including all user ID and password data.	
server.xml	config/cells/ *cell_name*/ nodes/ *node_name*/ servers/ *server_name*/	Identify a server and its components.	
serverindex.xml	config/cells/ *cell_name*/ nodes/ *node_name*/	Specify communication ports used on a specific node.	
spi.policy	config/cells/ *cell_name*/ nodes/ *node_name*/	Define security permissions for service provider libraries such as resource providers.	X
variables.xml	config/cells/ *cell_name*/ config/cells/ *cell_name*/ nodes/ *node_name*/ config/cells/ *cell_name*/ nodes/*node_name*/ servers/ *server_name*/	Configure variables used to parameterize any part of the configuration settings.	

`virtualhosts.xml`	`config/cells/`
	`cell_name``/`
		Configure a virtual host and its MIME types.	

Object names

When you create a new object using the administrative console or a wsadmin command, you often must specify a string for a name attribute. Most characters are allowed in the name string. However, the name string cannot contain the following characters. The name string also cannot contain leading and trailing spaces.

/	forward slash
\	backslash
*****	asterisk
,	comma
:	colon
;	semi-colon
=	equal sign
+	plus sign
?	question mark
|	vertical bar
<	left angle bracket
>	right angle bracket
&	ampersand (and sign)
%	percent sign
’	single quote mark
″	double quote mark
]]>	No specific name exists for this character combination.
.	period (not valid if first character; valid if a later character)

Configuration repositories

A configuration repository stores configuration data. By default, configuration repositories reside in the *config* subdirectory of the product installation root directory.

Handling temporary configuration files resulting from session timeout

If the console is not used for 15 minutes or more, the session times out. The same thing happens if you close the browser window without saving the configuration file. Changes to the file are saved to a temporary file when the session times out, after 15 minutes.

When a session times out, the configuration file in use is saved under the userid/timeout directory under the ServletContext’s temp area. This is the value of the javax.servlet.context.tempdir attribute of the ServletContext. By default, it is: *`install_root`*`/temp/hostname/Administration/admin/admin.war`

You can change the temp area by specifying it as a value for the tempDir init-param of the action servlet in the deployment descriptor (web.xml) of the administrative application.

The next time you log on to the console, you are prompted to load the saved configuration file. If you decide to load the saved file:

1. If a file with the same name exists in the *install_root*/config directory, that file is moved to the userid/backup directory in the temp area.
2. The saved file is moved to the *install_root*/config directory.
3. The file is then loaded.

If you decide not to load the saved file, it is deleted from the userid/timeout directory in the temp area.

The configuration file is also saved automatically when the same user ID logs into the non-secured console again, effectively starting a different session. This process is equivalent to forcing the existing user ID out of session, similar to a session timing out.

Changing the location of temporary configuration files

The configuration repository uses copies of configuration files and temporary files while processing repository requests. It also uses a backup directory while managing the configuration. You can change the default locations of these files from the configuration directory to a directory of your choice using system variables or the administrative console.

The default location for the configuration temporary directory is CONFIG_ROOT/temp. Change the location by doing either of the following:

- Set the system variable *was.repository.temp* to the location you want for the repository temporary directory. Set the system variable when launching a Java process using the -D option. For example, to set the default location of the repository temporary directory, use the following option:

  ```
  -Dwas.repository.temp=%CONFIG_ROOT%/temp
  ```
- Use the administrative console to change the location of the temporary repository file location for each server configuration. For example, on the Network Deployment product, to change the setting for a deployment manager, do the following:
  1. Click **System Administration > Deployment Manager** in the navigation tree of the administrative console. Then, click **Administration Services**, **Repository Service**, and **Custom Properties**.
  2. On the Properties page, click **New**.
  3. On the settings page for a property, define a property for the temporary file location. The key for this property is `was.repository.temp`. The value can include WebSphere Application Server variables such as `${WAS_TEMP_DIR}/config`. Then, click **OK**.

The system property set using the first option takes precedence over the configuration property set using the second option.

Changing the location of backed-up configuration files

During administrative processes like adding a node to a cell or updating a file, configuration files are backed up to a backup location. The default location for the backup configuration directory is CONFIG_ROOT/backup. Change the location by doing either of the following:

- Set the system variable *was.repository.backup* to the location you want as the repository backup directory. Set the system variable when launching a Java process using the -D option. For example, to set the default location of the repository backup directory, use the following option:

  ```
  -Dwas.repository.backup=%CONFIG_ROOT%/backup
  ```
- Use the administrative console to change the location of the repository backup directory for each server configuration. For example, on the Network Deployment product, do the following to change the setting for a deployment manager:
  1. Click **System Administration > Deployment Manager** in the navigation tree of the administrative console. Then, click **Administration Services**, **Repository Service**, and **Custom Properties**.
  2. On the Properties page, click **New**.
  3. On the settings page for a property, define a property for the backup file location. The key for this property is `was.repository.backup`. The value can include WebSphere Application Server variables such as `${WAS_TEMP_DIR}/backup`. Then, click **OK**.

The system property set using the first option takes precedence over the configuration property set using the second option.

Changing the location of temporary workspace files

The administrative console workspace allows client applications to navigate the configuration. Each workspace has its own repository location defined either in the system property or the property passed to a workspace manager when creating the workspace: workspace.user.root or workspace.root, which is calculated as %workspace.root%/*user_ID*/workspace/wstemp.

The default workspace root is calculated based on the user installation root: %user.install.root%/wstemp. You can change the default location of temporary workspace files by doing the following:

- Distributed platforms: Change the setting for the system variable *workspace.user.root* or *workspace.root* so its value is no longer set to the default location. Set the system variable when launching a Java process using the -D option. For example, to set the default location the full path of the root of all users' directories, use the following option:

  ```
  -Dworkspace.user.root=full_path_for_root_of_all_user_directories
  ```

Backing up and restoring administrative configurations

WebSphere Application Server represents its administrative configurations as XML files. You should back up configuration files on a regular basis.

1. Run the backupConfig command to back up configuration files.
2. Run the restoreConfig command to restore configuration files. Specify backup files that do not contain invalid or inconsistent configurations.

Transformation of configuration files

The WebSphere Application Server master configuration repository stores configuration files for all the nodes in the cell. When you upgrade the deployment manager from one release of WebSphere Application Server to another, the configuration files that are stored in the master repository for the nodes on the old release are converted into the format of the new release.

With this conversion, the deployment manager can process the configuration files uniformly. However, nodes on an old release cannot readily use configuration files that are in the format of the new release. WebSphere Application Server addresses the problem when it synchronizes the configuration files from the master repository to a node on an old release. The configuration files are first transformed into the old release format before they ship to the node. WebSphere Application Server performs the following transformations on configuration documents:

- Changes the XML name space from the format of the new release to the format of the old release
- Strips out attributes of cell-level documents that are applicable to the new release only
- Strips out new resource definitions that are not understood by old release nodes

Server configuration files: Resources for learning

Use the following links to find relevant supplemental information about administering WebSphere Application Server configuration files. 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:

Administration

- IBM WebSphere Application Server Redbooks at `http://publib-b.boulder.ibm.com/Redbooks.nsf/Portals/WebSphere`.

  This site contains a listing of all WebSphere Application Server Redbooks.
- IBM developerWorks WebSphere at `http://www.software.ibm.com/wsdd`.

  This site is the home of technical information for developers working with WebSphere products. You can download WebSphere software, take a fast path to developerWorks zones, such as VisualAge Java or WebSphere Application Server, learn about WebSphere products through a newcomers page, tutorials, technology previews, training, and Redbooks, get answers to questions about WebSphere products, and join the WebSphere community, where you can keep up with the latest developments and technical papers.
- WebSphere Application Server Support page at `http://www.ibm.com/software/webservers/appserv/support.html`.

  Take advantage of the Web-based Support and Service resources from IBM to quickly find answers to your technical questions. You can easily access this extensive Web-based support through the IBM Software Support portal at URL `http://www-3.ibm.com/software/support/` and search by product category, or by product name. For example, if you are experiencing problems specific to WebSphere Application Server, click **WebSphere Application Server** in the product list. The WebSphere Application Server Support page appears.

Administering application servers

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

This section describes how to create and configure application servers in an existing application server environment.

A WebSphere Application Server administrator can configure one or more application servers and perform tasks such as the following:

1. Create application servers.
2. Manage application servers.
3. Configure transport chains.
4. Develop custom services.
5. Define processes for the application server. As part of defining processes, you can define:
6. Use the Java virtual machine.

After preparing a server, deploy an application or component on the server. See “Preparing to host applications” on page 162 for a sample procedure that you might follow in configuring the application server runtime and resources.

Application servers

Application servers extend a Web server’s capabilities to handle Web application requests, typically using Java technology. An application server makes it possible for a server to generate a dynamic, customized response to a client request.

For example, suppose--

1. A user at a Web browser on the public Internet visits a company Web site. The user requests to use an application that provides access to data in a database.
2. The user request flows to the Web server.
3. The Web server determines that the request involves an application containing resources not handled directly by the Web server (such as servlets). It forwards the request to a WebSphere Application Server product.

4. The WebSphere Application Server product forwards the request to one of its application servers on which the application is running.
5. The invoked application then processes the user request. For example:
   - An application servlet prepares the user request for processing by an enterprise bean that performs the database access.
   - The application produces a dynamic Web page containing the results of the user query.
6. The application server collaborates with the Web server to return the results to the user at the Web browser.

The WebSphere Application Server product provides multiple application servers that can be either separately configured processes or nearly identical clones.

Creating application servers

For the Express and Base products, you must use scripting to create a new application server (see the *Administering applications and their environment* PDF). The server you create cannot be managed using the administrative console. Also note that the only server you can manage using the administrative console is the default server (server1).

With WebSphere Application Server Version 6.0, you can now upgrade a portion of the nodes in a cell, while leaving others at the older release level. This means that, for a period of time, you may be managing servers that are at the current release and servers that are running the newer release in the same cell. In this mixed environment, there are restrictions on what you can do with servers at the older release level. They are:

- You can only create new server definitions on nodes that are running WebSphere Application Server Version 6.0.
- When you create a new server definition, you must use a server configuration template, and that template must be created from a WebSphere Application Server Version 6.0 server instance. You cannot create (or use) a template from a WebSphere Application Server Version 5.x server instance.

There are no restrictions on what you can do with the servers running on the newer release level.

The steps below describe how to use the Create New Application Server page.

1. Create the new application server using the wsadmin **createApplicationServer** command. For information, see the *Administering applications and their environment* PDF.
2. To use multiple language encoding support in the administrative console, configure an application server with UTF-8 encoding enabled.

The new application server appears in the list of servers on the Application Servers page.

Note that the application server created has many default values specified for it. An application server has many properties that can be set and creating an application server on the Create New Application Server page specifies values for only a few of the important properties. To view all of the properties of your application server and to customize your application server further, click on the name of your application server on the Application Servers page and change the settings for your application server as needed.

Configuring application servers for UTF-8 encoding

To use multiple language encoding support in the administrative console, you must configure an application server with UTF-8 encoding enabled.

1. Create an application server or use an existing application server.
2. On the Application Server page, click on the name of the server you want enabled for UTF-8.
3. On the settings page for the selected application server, under Server Infrastructure, click **Java and Process Management > Process Definition**.

4. On the Process Definition page, click **Java Virtual Machine**.
5. On the Java Virtual Machine page, specify `-Dclient.encoding.override=UTF-8` for **Generic JVM Arguments** and click **OK**.
6. Click **Save** on the console task bar.
7. Restart the application server.

Note that the autoRequestEncoding option does not work with UTF-8 encoding enabled. The default behavior for WebSphere Application Server is, first, to check if charset is set on content type header. If it is, then the product uses content type header for character encoding; if it is not, then the product uses character encoding set on server using the system property default.client.encoding. If charset is not present and the system property is not set, then the product uses ISO-8859-1. Enabling autoRequestEncoding on a Web module changes the default behavior: if charset it not present on an incoming request header, the product checks the Accept-Language header of the incoming request and does encoding using the first language found in that header. If there is no charset on content type header and no Accept language header, then the product uses character encoding set on server using the system property default.client.encoding. As with the default behavior, if charset is not present and the system property is not set, then the product uses ISO-8859-1.

Managing application servers

To view information about an application server, use the Application Servers panel on the administrative console.

You must use the “Getting started with scripting” on page 186 to create a new application server. The server you create cannot be managed using the administrative console. Also note that the only server you can manage using the administrative console is the default server (server1).

1. Access the Application Servers page. Click **Servers > Application Servers** in the console navigation tree.
2. View information about application servers.

   The Application Servers page lists application servers in the cells holding the application servers.

   To view additional information about a particular application server or to further configure an application server, click on the application server name under **Name**. This accesses the settings page for an application server.

   To view product information for an application server:

   a. Verify that the application server is running.
   b. Display the **Runtime** tab on the settings page for an application server.
   c. Click **Product Information**.

   The Product Information page displayed lists the WebSphere Application Server products installed for the application server, the version and build levels for the products, the build dates, and any interim fixes applied to the application server.

   Note: You can also get this information by using the **versionInfo** command. For more information, see ″versionInfo command″ in the information center.
3. Create an application server using the wsadmin **createApplicationServer** command. For information, see “Commands for the AdminTask object” on page 489.
4. Monitor the running of application servers.

Server collection

Use this page to view information about and manage application servers, generic servers, Java Message Service (JMS) servers, and Web servers.

Application Servers

The Application Servers page lists the application servers in the cell. You can use this page to create new application servers, create application server templates, or delete existing application servers. You can also use this page to start and stop these application servers.

To view this administrative console page, click **Application Servers**.

Generic Servers

The Generic Servers page lists the generic servers in the cell. You can use this page to create new generic servers, create generic server templates, or delete existing generic servers. You can also use this page to start and stop these generic servers.

The Network Deployment product also shows the status of the generic servers. The status indicates whether a server is running, stopped, or encountering problems.

You can use this page to add or delete application servers.

To view this administrative console page, click **Generic Servers**.

Java Message Service (JMS) Servers

The JMS Servers page lists the JMS servers in the cell. You can use this page to start and stop these JMS servers.

Each JMS server provides the functions of the JMS provider for a node in your administrative domain. There can be at most one JMS server on each 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.

Note: JMS servers apply only to WebSphere Application Server Version 5.x nodes. You cannot create a JMS server on a node that is running WebSphere Application Server 6.0, but the existing Version 5.x JMS servers will continue to be displayed, and you can modify their properties. You can also delete Version 5.x JMS servers.

To view this administrative console page, click **JMS Servers**.

Web Servers

The Web Servers page lists the Web servers in your administrative domain. You can use this page to generate and propagate a Web server plug-in configuration file, create new Web servers, create new Web server templates, or delete existing Web servers. You can also use this page to start and stop these Web servers.

To view this administrative console page, click **Web Servers**.

Name:

Specifies a logical name for the server. For WebSphere Application Server, server names must be unique within a node.

Node:

Specifies the name of the node holding the server.

Version:

Specifies the version of the WebSphere Application Server product on which the server runs.

Status:

Indicates whether the server is started or stopped. (Network Deployment only)

Note that if the status is *Unavailable*, the node agent is not running in that node and you must restart the node agent before you can start the server.

Application server settings:

An application server is a server which provides services required to run enterprise applications. Use this page to view or change the settings of an application server instance.

To view this administrative console page, click **Servers > Application Servers >***server_name*.

On the **Configuration** tab, you can edit fields. On the **Runtime** tab, you can look at read-only information. The **Runtime** tab is available only when the server is running.

Name:

Specifies a logical name for the server. Server names must be unique within a node. However, for multiple nodes within a cluster, you may have different servers with the same server name as long as the server and node pair are unique.

For example, a server named *server1* in a node named *node1* in the same cluster with a server named *server1* in a node named *node2* is allowed. Configuring two servers named *server1* in the same node is not allowed. WebSphere Application Server uses the server name for administrative actions, such as referencing the server in scripting.

Data type	String
Default	server1

Run in development mode:

Enabling this option may reduce the startup time of an application server. This may include JVM settings such as disabling bytecode verification and reducing JIT compilation costs. Do not enable this setting on production servers. This setting is only available on application servers running WebSphere Application Server Version 6.0 and later.

Specifies that you want to use the JVM settings **-Xverify** and **-Xquickstart** on startup. After selecting this option, save the configuration and restart the server to activate development mode.

The default setting for this option is `false`, which indicates that the server will not be started in development mode. Setting this option to `true` specifies that the server will be started in development mode (with settings that will speed server startup time).

Data type	Boolean
Default	false

Parallel start:

Select this field to start the server on multiple threads. This might shorten the startup time.

Specifies that you want the server components, services, and applications to start in parallel rather than sequentially.

The default setting for this option is `true`, which indicates that the server be started using multiple threads. Setting this option to `false` specifies that the server will not be started in using multiple threads (which may lengthen startup time).

Note that the order in which the applications start depends on the weights you assigned to each them. Applications that have the same weight are started in parallel. You set an application's weight with the *Starting weight* option on the **Applications > Enterprise Applications >** *application_name* page of the Administrative Console. For more information about the *Starting weight* option, see the *Installing your application serving environment* PDF.

Data type	Boolean
Default	true

Class loader policy:

Select whether there is a single class loader to load all applications or a different class loader for each application.

Class loading 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 Developer Kit class loaders and WebSphere Application Server class loaders is `Parent first`.

If you select `Parent last`, your application can override classes contained in the parent class loader, but this action can potentially result in ClassCastException or linkage errors if you have mixed use of overridden classes and non-overridden classes.

Process Id:

The native operating system's process ID for this server.

The process ID property is read only. The system automatically generates the value.

Cell name:

The name of the cell in which this server is running.

The Cell name property is read only.

Node name:

The name of the node in which this server is running.

The Node name property is read only.

State:

The run-time execution state for this server.

The State property is read only.

Ports collection:

Use this page to view and manage communication ports used by run-time components running within a process. Communication ports provide host and port specifications for a server.

To view this administrative console page, click **Servers > Application Servers >***server_name* **Communications > Ports**.

Note that this page displays only when you are working with ports for application servers.

Port Name:

Specifies the name of a port. Each name must be unique within the server.

Host:

Specifies the IP address, domain name server (DNS) host name with domain name suffix, or just the DNS host name, used by a client to request a resource (such as the naming service, or administrative service).

Port:

Specifies the port for which the service is configured to accept client requests. The port value is used in conjunction with the host name.

Transport Details:

Provides a link to the transport chains associated with this port. If no transport chains are associated with this port, the string ″No associated transports″ appears in this column.

Ports settings:

Use this to view and change the configuration for a communication port used by run-time components running within a process. A communication port provides host and port specifications for a server.

For base WebSphere Application Server, you can view this administrative console page, by clicking **Servers > Application Servers >***server_name* **> Ports >***port_name*

Port Name:

Specifies the name of the port. The name must be unique within the server.

Note that this field displays only when you are defining a port for an application server. You can select a radio button to:

Well-known Port
: select a previously defined port from the drop down list

User-defined Port
: create a port with a new name by entering the name in the text box

Data type	String

Host:

Specifies the IP address, domain name server (DNS) host name with domain name suffix, or just the DNS host name, used by a client to request a resource (such as the naming service, administrative service, or JMS broker).

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`.

Host names on the ports can be resolvable names or IP addresses. The server will bind to the specific host name or IP address that is supplied. That port will only be accessible through the IP address that is resolved from the given host name or IP address. The IP address may be of the IPv4 (Internet Protocol Version 4) format for all platforms, and IPv6 (Internet Protocol Version 6) format on specific operating systems where the server supports IPv6.

Data type	String
Default	* (asterisk)

Port:

Specifies the port for which the service is configured to accept client requests. The port value is used in conjunction with the host name.

Port numbers in the server can be reused among multiple ports as long as they have host names that resolve to unique IP addresses and there is not a port with the same port number and a wildcard (*) host name. A port number is valid in the range of 0 and 65535. 0 specifies that the server should bind to any ephemeral port available.

Data type	Integer
Default	None
Range	1-65536

Custom property collection:

Use this page to view and manage arbitrary name-value pairs of data, where the name is a property key and the value is a string value that can be used to set internal system configuration properties.

The administrative console contains several Custom Properties pages that work similarly. To view one of these administrative pages, click a **Custom Properties** link.

Name:

Specifies the name (or key) for the property.

Do not start your property names with `was.` because this prefix is reserved for properties that are predefined in WebSphere Application Server.

Value:

Specifies the value paired with the specified name.

Description:

Provides information about the name-value pair.

Custom property settings:

Use this page to configure arbitrary name-value pairs of data, where the name is a property key and the value is a string value that can be used to set internal system configuration properties. Defining a new property enables you to configure a setting beyond that which is available in the administrative console.

For base WebSphere Application Server you can view this administrative console page by, clicking **Servers > Application Servers >***server_name*. Then, under Server Infrastructure, click **Administration** > **Custom Properties**

Name:

Specifies the name (or key) for the property.

Do not start your property names with `was.` because this prefix is reserved for properties that are predefined in WebSphere Application Server.

Data type	String

Value:

Specifies the value paired with the specified name.

Data type	String

Description:

Provides information about the name and value pair.

Data type	String

Server component collection:

Use this page to view information about and manage server component types such as application servers, messaging servers, or name servers.

To view this administrative console page, click **Servers > Application Servers >***server_name*. Then, under Server Infrastructure, click **Administration > Server Components**.

Type:

Specifies the type of internal server.

Server component settings:

Use this page to view or configure a server component instance.

To view this administrative console, click **Servers > Application Servers >***server_name*. Then, under Server Infrastructure, click **Administration > Server Components >***server_component_name*.

Name:

Specifies the name of the component.

Data type	String

Initial State:

Specifies the desired state of the component when the server process starts. The options are: *Started* and *Stopped*. The default is *Started*.

Data type	String
Default	Started

Thread pool collection:

Use this page to select or create a group of threads that an application server uses. Requests are sent to the server through any of the HTTP transports. A thread pool enables components of the server to reuse threads to eliminate the need to create new threads at run time. Creating new threads expends time and resources.

To view this administrative console page, click **Servers > Application Servers >***server_name* **> Thread Pools**. (You can reach this page through more than one navigational route.)

Thread pool settings:

Use this page to configure a group of threads that an application server uses. Requests are sent to the server through any of the HTTP transport channels or HTTP transports. A thread pool enables components of the server to reuse threads to eliminate the need to create new threads at run time. Creating new threads expends time and resources.

To view this administrative console page, click **Servers > Application Servers >***server_name* **> Thread Pools**, then select the thread pool. (You can reach this page through more than one navigational route.)

Minimum size:

Specifies the minimum number of threads to allow in the pool.

Data type	Integer
Default	10

Maximum size:

Specifies the maximum number of threads to allow in the pool.

If your Tivoli Performance Viewer shows the Percent Maxed metric to remain consistently in the double digits, consider increasing the Maximum size. The Percent Maxed metric indicates the amount of time that the configured threads are used. If there are several simultaneous clients connecting to the server-side ORB, increase the size to support up to 1000 clients.

Data type	Integer
Default	50
Recommended	50 (25 on Linux systems)

Thread inactivity timeout:

Specifies 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 *server.xml* file.

Data type	Integer
Units	Milliseconds
Default	3500

Allow thread allocation beyond maximum thread size:

Specifies whether the number of threads can increase beyond the maximum size configured for the thread pool.

Data type	Boolean
Default	Not enabled (false)

Generic server settings:

Use this page to view or change the settings of a generic server.

A generic server is a server that is managed in the WebSphere Application Server 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, including a Java server, a C or C++ server or process, or a Remote Method Invocation (RMI) server.

To view this administrative console page, click **Servers > Generic Servers >***server_name*.

On the **Configuration** tab, you can edit fields. On the **Runtime** tab, you can look at read-only information. The **Runtime** tab is available only when the server is running.

Name:

Specifies a logical name for the generic server.

It is highly recommended that you use a naming scheme that makes it easy to distinguish your generic application servers from regular WebSphere Application Servers. This will enable you to quickly determine whether to use the Terminate or Stop button in the administrative console to stop a specific application server.

You must use the Terminate button to stop a generic application server.

Data type	String
Default	

Starting servers

Starting a server starts a new server process based on the process definition settings of the current server configuration.

If you need to restart a server, follow the directions in this article for starting servers. The procedure that applies to starting servers also applies to restarting servers.

Note: If you created a new server definition using a base WebSphere Application Server, you cannot start, stop, or manage the new server using the original base Application Server.

There are several options for starting an Application Server:

- Windows If you are using Windows, you can use the **Start** menu to start the Application Server. If you are using the Express version of the product, click **Start > Programs > IBM WebSphere > Express v6.0 > Start the server**. You can check that the server has successfully started by checking the `startServer.log` file. If the server has successfully started, the last two lines of the `startServer.log` file reads:

  ```
  Server launched.  Waiting for initialization status.
  Server server1 open for e-business; process id is 1932.
  ```

The `startServer.log` file is located in the `<drive>:\Program Files\IBM\WebSphere\AppServer\profiles\`*`profile_name`*`\logs\server1` directory if you have installed your server with the default settings. The server name and process id vary depending on your settings.

- On distributed platforms, use the **startServer** command to start an Application Server from the command line.
- AIX On AIX, you can use the command line to start the server. Use the **startServer** command from the `/usr/WebSphere/AppServer/bin` directory, as shown below. To start a server that is associated with a non-default profile, issue the **startServer** command from the `/opt/WebSphere/AppServer/profiles/profile_name/bin` directory.

  ```
  #  ./startServer.sh server1
  ```

  You can check that the server has successfully started by checking the `startServer.log` file. If the server has successfully started, the last two lines of the `startServer.log` file reads:

  ```
  Server launched.  Waiting for initialization status.
  Server server1 open for e-business; process id is 1932.
  ```

  On AIX, the startServer.log file is located in the `/usr/IBM/WebSphere/AppServer/profiles/`*`profile_name`*`/logs/server1/` directory.
- Start an Application Server for tracing and debugging.

  To start the Application Server with standard Java debugging enabled:

  1. Click **Servers > Application Servers** from the administrative console navigation tree. Then, click the Application Server whose processes you want to trace and debug, then **Java and Process Management > Process Definition > Java Virtual Machine**.
  2. On the Java Virtual Machine page, place a checkmark in the check box for the **Debug Mode** setting to enable the standard Java debugger. If needed, set debug arguments. Then, click **OK**.
  3. Save the changes to a configuration file.
  4. Stop the Application Server.
  5. Start the Application Server again as described previously.

Once the server is started, you can install your applications.

Running application servers from a non-root user

By default, each base WebSphere Application Server server on a Linux and UNIX platform uses the root user ID to run all application server processes. However, you can run all application server processes under the same non-root user and user group. This task describes how to run an application server process from a non-root user.

If global security is enabled, the user registry must not be Local OS. Using the Local OS user registry requires the application server to run as root. Refer to the *Securing applications and their environment* PDF for details.

Run your application servers as non-root when you no longer want to use root authority. For security or administrative reasons, you may want to change to non-root user IDs. Perform this task at any time to change the permissions of an application server. You must restart the application server in order for the changes to take effect. .

Note: If you are using the Tivoli Access Manager (TAM) to perform authentication or authorization for WebSphere Application Server, it is important to be aware of potential permissions problems. For more information, see the *Securing applications and their environment* PDF.

For the following steps, assume that:

- `was1` is the user to run the application server
- `wasgroup` is the primary user group for user was1

- `wasnode` is the node name
- `server1` is the application server
- `/opt/WebSphere/AppServer` is the installation root
- `nodeProfile1` is the profile name.

Note: For information about creating a profile, see the *Administering applications and their environment* PDF.

To configure an application server to run as non-root, complete the following steps.

1. Log on to the application server system as the root user.
2. Create the user ID `was1` with a primary user group of `wasgroup`. The user ID, `was1`, is an example. You can name the user something else.
3. Log off and back on as root.
4. Start server1 as root. Run the `startServer.sh` script from the `/bin` directory of the installation root:

   ```
   startServer.sh server1
   ```
5. Specify user and group ID values for the **Run As User** and **Run As Group** settings for a server:
   a. Start the administrative console.
   b. Go to the Process execution page of the administrative console. You must define all three properties in the following table. Click **Servers > Application Servers > server1 > Server Infrastructure > Java and Process Management > Process Execution** and change all of the following values:

Property	Value
Run As User	was1
Run As Group	wasgroup
UMASK	022

   c. Click **OK**.
   d. Save the configuration.
6. Stop the application server. Use the `stopServer.sh` script from the `/bin` directory of the installation root:

   ```
   stopServer.sh server1
   ```
7. Change file permissions as the root user. The following example assumes that the installation root directory for WebSphere Application Server is `/opt/WebSphere/AppServer`:

   ```
   chgrp wasgroup /opt/WebSphere
   chgrp wasgroup /opt/WebSphere/AppServer
   chgrp -R wasgroup /opt/WebSphere/AppServer/cloudscape
   chgrp -R wasgroup /opt/WebSphere/AppServer/profiles/nodeProfile1
   chmod g+wr /opt/WebSphere
   chmod g+wr /opt/WebSphere/AppServer
   chmod -R g+wr /opt/WebSphere/AppServer/cloudscape
   chmod -R g+wr /opt/WebSphere/AppServer/profiles/nodeProfile1
   ```
8. Log on to the application server system as `was1`.
9. Start server1 as `was1`. Run the `startServer.sh` script from the `/bin` directory of the installation root:

   ```
   startServer.sh server1
   ```
10. If creating another server with a different user ID, follow this procedure again for the new user ID and server name.

    The two user IDs must share the same group, `wasgroup`.

You can start an application server from a non-root user.

Detecting and handling problems with run-time components

You must monitor the status of run-time components to ensure that, once started, they remain operational as needed.

1. Regularly examine the status of run-time components. Browse messages displayed under **Websphere Runtime Messages** in the status area at the bottom of the console. The run-time event messages marked with a red **X** provide detailed information on event processing. You can also use the Logging and Tracing page of the administrative console to monitor the status of run-time components. Click **Troubleshooting > Logs and Trace** in the console navigation tree to access the page.
2. If an application stops running when it should be operational, examine the application's status on an Applications page and try restarting the application.
3. If the run-time components do not restart, reexamine the messages and read information on problem determination to help you to restart the components.

Stopping servers

Stopping an application server stops a server process based on the process definition settings in the current application server configuration.

- Windows In Windows, you can use the Start menu to stop your application server. Click **Start > Programs > IBM WebSphere > Express v6.0 > Stop the server**. When the server stops successfully, the stopServer.log file contains the following in the last two lines:

  ```
  Server stop request issued.  Waiting for stop status.
  Server server1 stop completed.
  ```

  The server name varies depending on your settings.
- Use the **stopServer** command to stop an application server from the command line.

  A warning message appears if you are stopping the application server that is running the administrative console application.
- Stop the application server from the command line. In distributed environments, you can use the **stopServer** command to stop a single server. In AIX, use the **stopServer** or the **stopManager** command from the `/usr/WebSphere/AppServer/bin` directory:

  ```
  #  ./stopServer.sh server1
  ```

  ```
  #   ./stopManager.sh
  ```

Creating generic servers

There are two types of generic application servers:
- Non-Java applications or processes.
- Java applications or processes

You can use the wsadmin tool or the **Generic servers** panel of the administrative console to create either type.

Note: For the Base WebSphere Application Server product, although you can use the administrative console to create a generic application server definition, you cannot use it to start, stop or, in any way, control or manage that application server. The Base product administrative console can only be used to create server definitions and, if necessary, adjust the server definitions that it creates. To manage Base generic application servers, use the wsadmin tool.

- **Create a non-Java application as a generic server.** The following steps describe how to use the administrative console to create a non-Java application as a generic application server.
  1. Select **Servers** > **Generic servers**
  2. Click **New**. You can then specify the name of the generic server you are creating.

3. Type in a name for the generic server. The name must be unique within the application server. It is highly recommended that you use a naming scheme that makes it easy to distinguish your generic application servers from regular Websphere Application Servers. This will enable you to quickly determine whether to use the **Terminate** or **Stop** button in the administrative console to stop specific application server. You must use the **Terminate** button to stop a generic application server.
4. Select a template to use in creating the new server. You can use a default application server template for your new server or use an existing application server as a template. The new application server will inherit all properties of the template server. If you create the new server using an existing application server do not enable the option to map applications from the existing server to the new server. This option does not apply for a generic server.
5. Click **Next**
6. Click **Finish**. The generic server now appears as an option on the **Generic servers** panel in the administrative console.
7. On the **Generic servers** panel, click on the name of the generic server.
8. Under **Additional Properties** click **Process Definition**.
9. In the **Executable name** field under **General Properties**, enter the name of the non-WebSphere Application Server program that is to be launched when you start this generic server. Executable target type and Executable target properties are not used for non-Java applications. Executable target type and Executable target properties are only used for Java applications
10. Click **OK**.

- **Create a Java application as a generic server:** The following steps describe how to use the administrative console to create a Java application as a generic application server.
  1. Select **Servers** > **Generic servers**
  2. Click **New**. You can then specify the name of the generic server you are creating.
  3. Type in a name for the generic server. The name must be unique within the application server. It is highly recommended that you use a naming scheme that makes it easy to distinguish your generic application servers from regular Websphere Application Servers. This will enable you to quickly determine whether to use the **Terminate** or **Stop** button in the administrative console to stop specific application server. You must use the **Terminate** button to stop a generic application server.
  4. Click **Next**
  5. Click **Finish**. The generic server now appears as an option on the **Applications Server** panel in the administrative console.
  6. Click **Finish**. The generic server now appears as an option on the **Generic servers** panel in the administrative console.
  7. On the **Generic servers** panel, click on the name of the generic server.
  8. Under **Additional Properties** click **Process Definition**.
  9. In the **Executable name** field under **General Properties**, enter the path for Websphere Application Server's default JVM (${JAVA_HOME}/bin/java), which will be used to run the Java application when you start this generic server.
  10. In the **Executable target type** field under **General Properties**, select whether a Java class name, **JAVA_CLASS**, or the name of an executable JAR file, **EXECUTABLE_JAR**, will be used as the executable target of this Java process. The default for Websphere Application Server is **JAVA_CLASS**.
  11. In the **Executable target** field under **General Properties**, enter the name of the executable target. (Depending on the executable target type, this will be either a Java class containing a main() method, or the name of an executable JAR file.) The default for Websphere Application Server is **com.ibm.ws.runtime.WsServer**.
  12. Click **OK**.

Note: If the generic server is to run an application server other than the WebSphere Application Server, leave the **Executable name** field set to the default value and specify the Java class containing the main function for your application serve in the **Executable target** field.

You can now start and terminate the generic server whenever you want to start or terminate the non-WebSphere Application Server server or process associated with this server.

Starting and terminating generic servers

This topic describes how to start and terminate generic servers.

If you created a generic server on a Base WebSphere Application Server, you cannot start, terminate, or monitor this server with the Base Application Server administrative console. You must use the wsadmin tool to manage Base generic servers.

Starting generic servers

There are two ways to start a generic server in a Network Deployment environment:

- Use the administrative console:
  1. From the administrative console navigation tree, select **Servers > Application Servers**.
  2. Select the check box beside the name of the generic server, and then click **Start**.
  3. View the **Status** value and any messages or logs to see whether the generic server starts.
- Use the MBean NodeAgent launchProcess operation of the wsadmin tool.

Terminating generic servers

There are two ways to terminate a generic server in a Network Deployment environment:

- Use the administrative console:
  1. From the administrative console navigation tree, select **Servers > Application Servers**.
  2. Select the check box beside the name of the generic server, and then click **Terminate**.
  3. View the **Status** value and any messages or logs to see whether the generic server terminates.

  Note: The **Stop** and **Stop Immediate** buttons on the administrative console do not work for generic servers.
- Use the MBean terminate launchProcess operation of the wsadmin tool.

Configuring transport chains

You need to 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.

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

1. Create a transport chain: You can either use the administrative console or wsadmin commands to create a transport chain. If you want to use the administrative console:
   a. Ensure that a port is available for the new transport chain.
   b. In the administrative console, click **Servers > Application servers >***server_name*, and then click on one of the following:
      - Under **Web container settings**, click **Web container transport chains**.

- Under **Server messaging**, click either **Messaging engine inbound transports** or **WebSphere MQ link inbound transports**.

c. Click **New**. The Create New Transport Chain wizard initializes. During the transport chain creation process, you are asked to:

- Specify a name for the new chain.
- Select a transport chain template
- Select a port, if one is available to which the new transport chain will be bound. If a port is not available or you want to define a new port, specify a port name, the host name or IP address for that port, and a valid port number.

When you click **Finish**, the new transport chain is added to the list of defined transport chains on the **Transport chain** panel.

2. Click on the transport chain's name to view the configuration settings that are in effect for the transport channels contained in this chain. To change any of these settings:
   a. Click on the channel that requires changes to its settings.
   b. Make your changes to the configuration settings. Some of the settings, such as the port number are determined by what is specified for the transport chain when it is created and cannot be changed.
   c. Click on **Custom properties** to set any custom properties that have been defined for your system.
3. When you have made all of your changes, click **OK**.
4. Stop the application server and start it again. You must stop the application server and start it again before the configuration changes you made take affect.

Transport chains

Transport chains represent a network protocol stack that is used for I/O operations within an application server environment. Transport chains are part of the channel framework function that provides a common networking service for all components, including the service integration bus component of IBM service integration technologies, WebSphere Secure Caching Proxy, and the high availability manager core group bridge service.

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

The transport chain configuration settings determine which I/O protocols are supported for that chain. Following are some of the more common types of channels. Custom channels that support requirements unique to a particular customer or environment can also be added to a transport chain.

TCP channel

Used to provide client applications with persistent connections within a Local Area Network (LAN). When configuring a TCP channel, you can specify a list of IP addresses that are allowed to make inbound connections and a list of IP addresses that are not allowed to make inbound connections. You can also specify the thread pool that this channel uses, which allows you to segregate work by the port that the application server is listening on.

HTTP channel

Used to enable communication with remote servers. It implements the HTTP 1.0 and 1.1 standards and is used by other channels, such as the Web container channel, to server HTTP requests and to send HTTP specific information to servlets expecting this type of information.

HTTP Tunnel channel

Used to provide client applications with persistent HTTP connections to remote hosts that are either blocked by firewalls or require an HTTP proxy server (including authentication) or both. An HTTP Tunnel channel enables the exchange of application data in the body of an HTTP request or response that is sent to or received from a remote server. An HTTP Tunnel channel also enables client-side applications to poll the remote host and to use HTTP requests to either send data from the client or to receive data from an application server. In either case, neither the client nor the application server is aware that HTTP is being used to exchange the data.

Web container channel
Used to create a bridge in the transport chain between an HTTP inbound channel and a servlet and JavaServer Pages (JSP) engine.

DCS channel
Used by the core group bridge service, the data replication service (DRS), and the high availability manger to transfer data, objects, or events among application servers.

MQ channel
Used in combination with other channels, such as a TCP channel, within the confines of WebSphere MQ support to facilitate communications between a WebSphere System Integration Bus and a WebSphere MQ client or queue manager.

JFAP channel
Used by the Java Message Service (JMS) server to create connections to JMS resources on a service integration bus.

SSL channel
Used to associate an SSL configuration repertoire with the transport chain. This channel is only available when Secure Sockets Layer (SSL) support is enabled for the transport chain. An SSL configuration repertoire is defined in the administrative console, under security, on the **SSL configuration repertoires > SSL configuration repertoires** page.

HTTP transport channel custom property

If you are using an HTTP transport channel, you can add the following custom property to the configuration settings for that HTTP transport channel.

To add a custom property:

1. In the administrative console, click **Application servers >** *server_name* **Web container settings > Web container transport chains** >*chain_name* **> HTTP Inbound Channel > Custom Properties > New**
2. Under **General Properties** specify the name of the custom property in the Name field and a value for this property in the Value field. You can also specify a description of this property in the Description field.
3. Click **Apply** or **OK**.
4. Click **Save** to save your configuration changes.
5. Restart the server.

Following is a list of custom properties provided with the application server. These properties are not shown on the settings page for an HTTP transport channel.

inProcessLogFilenamePrefix

Use to specify a prefix for the filename of the network log file. Normally, when inprocess optimization is enabled, requests through the inprocess path are logged based on the logging attributes set up for the Web container's network channel chain. You can use this property to add a prefix to the filename of the network log file. This new filename is then used as the filename for the log file for inprocess requests. Requests sent through the inprocess path are logged to this file instead of to the network log file. For example, if the log file for a network transport chain is named `.../httpaccess.log`, and this property is set to `local` for the HTTP channel in that chain, the filename of the log file for inprocess requests to the host associated with that chain is `.../localhttpaccess.log`.

Data type	String

HTTP Tunnel transport channel custom property

If you are using an HTTP Tunnel transport channel, you can add the following custom property to the configuration settings for that HTTP Tunnel transport channel.

To add a custom property:

1. In the administrative console, click **Servers > Application servers >***server_name* **> Ports**. Click on **View associated transports** for the HTTP Tunnel port to whose configuration settings you want to add this custom property.
2. Click **New**.
3. Under **General Properties** specify the name of the custom property in the Name field and a value for this property in the Value field. You can also specify a description of this property in the Description field.
4. Click **Apply** or **OK**.
5. Click **Save** to save your configuration changes.
6. Restart the server.

Following is a description of the custom property that is provided with the application server. This property is not shown on the settings page for an HTTP Tunnel transport channel.

pluginConfigurable

Indicates whether or not the configuration settings for the HTTP Tunnel transport channel are included in the plugin-cfg.xml file for the Web server associated with the application server that is using this channel. Configuration settings for each of the Web container transport channels defined for an application server are automatically included in the plugin-cfg.xml file for the Web server associated with that application server.

Data type	Boolean
Default	False

Troubleshooting transport chain problems

TCP transport channel fails to bind to a specific host/port combination

If a TCP transport channel fails to bind to a specific port, one of the following situations might have occurred:

- You are trying to bind the channel to a port that is already bound to another application, such as another instance of a WebSphere Application Server.
- You are trying to bind to a port that is in a transitional state waiting for closure. This socket must transition to closed before you restart the server. The port might be in TIME_WAIT, FIN_WAIT_2, or CLOSE_WAIT state. Issue the `netstat -a` command from a command prompt window to display the state of the port to which you are trying to bind. If you need to change the amount of elapse time that must occur before TCP/IP can release a closed connection and reuse its resources, see the *Troubleshooting and support* PDF.

Configuring HTTP transports

Important: On a distributed platform, HTTP transport support is deprecated. Therefore, the administrative console page used to configure an HTTP transport is not available unless your migrated an HTTP transport from your V5 environment. You must define an HTTP transport channel instead of an HTTP transport to handle your HTTP requests.

An HTTP transport is the request queue between a WebSphere Application Server plug-in for Web servers and a Web container in which the Web modules of an application reside. To define the characteristics of the connections between that plug-in and the Web container, you must specify:

- How the transport is to handle a set of connections. For example, you must specify the number of concurrent requests that is to be allowed.
- Whether to secure the connections with SSL.
- The Host and IP information for the transport participants.

1. Change the configuration for an existing HTTP transport.
   a. Ensure that virtual host aliases include port values for the transport your are changing.
   b. Go to the HTTP Transports page and click on the transport under **Host** whose configuration you want to change.

      Remember, on a distributed platform, HTTP transport support is deprecated. Therefore you cannot view this administrative console page unless you are a V5.x user who, during the V6 migration process, indicated that you want to continue using the HTTP transports that are defined for your V5 environment.
   c. On the settings page for an HTTP transport, which might have the page title DefaultSSLSettings, change the specified values as needed, then click **OK**.
   d. Custom properties page, add and set any custom properties you want to use.
2. Stop the WebSphere Application Server and start it again. You must stop the WebSphere Application Server and start it again before the configuration changes you made take affect.

If the Web server is located on a machine remote from the Application Server, copy the `plugin-cfg.xml` file to the remote Web server and replace the file that is there. See ″Editing Web server configuration files″ in the information center for information about copying the `plugin-cfg.xml` and binary plug-in module to a remote Web server and configuring the Web server to use the files.

HTTP transport collection:

Use this page to view or manage HTTP transports. Transports provide request queues between WebSphere Application Server plug-ins for Web servers and Web containers in which the Web modules of applications reside. When you request an application in a Web browser, the request is passed to the Web server, then along the transport to the Web container.

On a distributed platform, if, when migrating from WebSphere Application Server Version 5.x, you indicate that you want to continue using an HTTP transport to handle your HTTP requests, your Version 5.x transports are migrated for you. If you are not migrating from Version 5.x, you must set up an HTTP transport channel to handle your HTTP requests.

To view the HTTP Transport administrative console page, click **Servers > Application Servers >***server_name* **> Web Container Settings > Web Container > HTTP Transports**.

the HTTP Transport panel on the administrative console

Host:

Specifies the host IP address to bind for transport. If the application server is on a local machine, the host name might be `localhost`.

Port:

Specifies the port to bind for transport. The port number can be any port that currently is not in use on the system. The port number must be unique for each application server instance on a given machine.

For distributed platforms, there is no limit to the number of HTTP ports that are allowed per process.

SSL Enabled:

Specifies whether to protect connections between the WebSphere plug-in and application server with Secure Sockets Layer (SSL). The default is not to use SSL.

HTTP transport settings:

Use this page to view and configure an HTTP transport. The name of the page might be that of an SSL setting such as DefaultSSLSettings.

On a distributed platform, if, when migrating from Version 5, you indicate that you want to continue using an HTTP transport to handle your HTTP requests, your Version 5 transports are migrated for you. If you are not migrating from WebSphere Application Server Version 5.x, you must set up an HTTP transport channel to handle your HTTP requests.

To view the HTTP Transport panel on the administrative console, click **Servers** > **Application Servers** >*server_name* > **Web Container Settings** > **Web Container** > **HTTP Transports** >*host_name*.

Host:

Specifies the host IP address to bind for transport.

If the application server is on a local machine, the host name might be `localhost`.

Data type	String

Port:

Specifies the port to bind for transport. Specify a port number between 1 and 65535. The port number must be unique for each application server on a given machine.

Data type	Integer
Range	1 to 65535

SSL Enabled:

Specifies whether to protect connections between the WebSphere Application Server plug-in and application server with Secure Sockets Layer (SSL). The default is not to use SSL.

Data type	Boolean
Default	false

SSL:

Specifies the Secure Sockets Layer (SSL) settings type for connections between the WebSphere Application Server plug-in and application server. The options include one or more SSL settings defined in the Security Center; for example, DefaultSSLSettings, ORBSSLSettings, or LDAPSSLSettings.

Data type	String
Default	An SSL setting defined in the Security Center

Transports:

A transport is the request queue between a WebSphere Application Server plug-in for Web servers and a Web container in which the Web modules of an application reside. When a user at a Web browser requests an application, the request is passed to the Web server, then along the transport to the Web container.

Transports define the characteristics of the connections between a Web server and an application server, across which requests for applications are routed. Specifically, they define the connection between the Web server plug-in and the Web container of the application server.

Administering transports is closely related to administering WebSphere Application Server plug-ins for Web servers. Indeed, without a plug-in configuration, a transport configuration is of little use.

On a distributed platform, when migrating from WebSphere Application Server Version 5.x, you indicate that you want to continue using an HTTP transport to handle your HTTP requests, your Version 5.x transports are migrated for you. If you are not migrating from Version 5.x, you must set up an HTTP transport channel to handle your HTTP requests.

The internal transport

The internal HTTP transport allows HTTP requests to be routed to the application server directly through a Web server plug-in. Logging is provided for debug purposes.

Prior to WebSphere Application Server Version 5.0.2, the HTTP transport functionality existed only as a means of accepting HTTP requests forwarded by an HTTP plug-in that was connected to a Web server. In WebSphere Application Server Version 5.0.2, HTTP transport functionality is now a supported internal Web server. By default, the internal HTTP transport listens for HTTP requests on port 9080 and for HTTPS requests on port 9443.

For example, use the URL `http://localhost:9080/snoop` to send requests to the snoop servlet on the local machine over HTTP and `https://localhost:9443/snoop` to send requests to the snoop servlet on the local machine over HTTPS.

The transport configuration is a part of the Web container configuration. You can configure the internal transport to use ports other than 9080 and 9443. However, you must also adjust your virtual host alias and what you type into the Web browser.

HTTP transport custom properties:

Use this page to set custom properties for an HTTP transport.

On a distributed platform, HTTP transport support is deprecated. Therefore you cannot create a new HTTP transport. Instead you must create an HTTP transport channel to handle your HTTP requests. If you are a WebSphere Application Server Version 5.x user who has migrated to Version 6, and during the migration process you indicated that you want to continue using an HTTP transport to handle your HTTP requests, your Version 5.x transports are still available for your use.

If you are using HTTP transports, you can set the following custom properties on either the Web Container or HTTP Transport **Custom Properties** panel on the administrative console. When set on the Web container **Custom Properties** page, all transports inherit the properties. Setting the same properties on a transport overrides like settings defined for a Web container.

To specify values for these custom properties for a specific transport on the HTTP Transport **Custom Properties** page:

1. In the console navigation tree, click **Servers > Application Servers >***server_name* **> Web Container settings > Web Container >HTTP Transport**

To specify a custom property:

1. Click on the **HOST** whose properties you want to set.
2. Under **Additional Properties** select **Custom Properties**.
3. On the Custom Properties page, click **New**.
4. 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.
5. Click **Apply** or **OK**.
6. Click **Save** on the console task bar to save your configuration changes.
7. Restart the server.

Following is a list of custom properties provided with the Application Server. These properties are not shown on the settings page for an HTTP transport.

ConnectionIOTimeOut:

Use the `ConnectionIOTimeOut` property to specify the maximum number of seconds to wait when trying to read or process data during a request.

Data type	Integer
Default	For distributed platforms: 5 seconds

ConnectionKeepAliveTimeout:

Use the `ConnectionKeepAliveTimeout` property to specify the maximum number of seconds to wait for the next request on a keep alive connection.

Data type	Integer

MaxConnectBacklog:

Use the `MaxConnectBacklog` property to specify the maximum number of outstanding connect requests that the operating system will buffer while it waits for the application server to accept the connections. If a client attempts to connect when this operating system buffer is full, the connect request will be rejected.

Set this value to the number of concurrent connections that you would like to allow. Keep in mind that a single client browser might need to open multiple concurrent connections (perhaps 4 or 5); however, also keep in mind that increasing this value consumes more kernel resources. The value of this property is specific to each transport.

Data type	Integer
Default	511

MaxKeepAliveRequests:

Use the `MaxKeepAliveRequests` property to specify the maximum number of requests which can be processed on a single keep alive connection. This parameter can help prevent denial of service attacks when a client tries to hold on to a keep-alive connection. The Web server plug-in keeps connections open to the application server as long as it can, providing optimum performance.

Data type	Integer
Default	For distributed platforms: 100 requests

KeepAliveEnabled:

This property is only valid in a distributed environment. Use the `KeepAliveEnabled` property to specify whether or not to keep connections alive

Data type	String
Default	true

Trusted:

This property is only valid in a distributed environment. Use the `Trusted` property to indicate that the application server can use the private headers that the Web server plug-in adds to requests.

Data type	String
Default	false

Configuring error logging for internal Web server HTTP transport:

To debug potential problems with using the HTTP transport as an internal Web server, you can use the following error logging capabilities.

1. Turn error logging on. To turn error logging on, add the following custom property to the HTTP Transport's configuration settings, and set the value to **false**:

   ```
   Property name: ErrorLogDisable
   Value: True/False
   Default: Error log is disabled by default
   ```

   When you are ready to turn error logging off, set the value of the **ErrorLogDisable** property back to **true**.
2. To specify your own error log file, add the following property to the transport section of the `server.xml` file:

   ```
   Property name: ErrorLog
   Value: <filename>
   Default: logs/<server instance>/http.log
   ```

   The error log property is used to specify where to place the error log. For example:<properties xmi:id=″WebContainer_Property_6″ name=″ErrorLog″ value=″logs/<server instance>/http.log″/>

   Note: The error log should appear in each instance of the server.

   If you are going to be using error logging for multiple HTTP transports in a single HTTP server, make sure you specify a unique filename for the error log file associated with each HTTP transport.
3. Add the LogLevel property to the transport section of the `server.xml` file to specify the level of messages to log in the error log file.

   ```
   Property name: LogLevel
   Value:  <level> (Levels include: debug, info, warn, error, crit)
   Default:  warn (warn includes error and crit; debug includes all levels)
   Scope:  Virtual/Global
   ```

   Log levels specify the type of message that appears in the error log. The *warn*, *error*, and *crit* messages are logged by default.
4. Restart the server.

If you have enabled error logging and encounter an error, there should be an error log message in the error log file you specified.

Configuring access logging for internal Web server HTTP transport:

To debug potential problems with using the HTTP transport as an internal Web server, you can use the following access logging capabilities.

1. Turn access logging on. To turn access logging on, add the following custom property to the HTTP Transport's configuration settings, and set the value to **false**:

   ```
   Property name: AccessLogDisable
   Values: True/False
   Default: Access log is disabled by default
   ```

   When you are ready to turn access logging off, set the value of the **AccessLogDisable** property back to **true**.
2. To specify your own access log file, add the following property to the transport section of the `server.xml` file:

   ```
   Property name: AccessLog
   Value: <filename>
   Default Value: logs/<server instance>/http_access.log
   ```

   The default access log file is `logs/<server_instance>/http_access.log`. Access log entries should have the format:

   ```
   <hostname or IP> <user agent> [<local time> -<status code>] <thread id> <http request> <status code> <bytecount>
   ```

   Note: If you are going to be using access logging for multiple HTTP transports in a single HTTP server, make sure you specify a unique filename for the access log file associated with each HTTP transport.
3. Restart the server.

If you have enabled access logging, there will be an access log in the location you specified.

Transport chains collection

Use this page to view or manage transport chains. Transport chains enable communication through transports, or protocol stacks, which are usually socket based.

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

The **Transport chains** page lists the transport chains defined for the selected application server. Transport chains represent network protocol stacks operating within this application server.

To view this administrative console page, click **Servers > Application servers >** *server_name* **> Ports**. Click on **View associated transports** for the port whose transport chains you want to view.

Name: Specifies a unique identifier for the transport chain. For WebSphere Application Server, transport name must be unique within a WebSphere Application Server configuration. Click on the name of a transport chain to change its configuration settings.

Enabled: When set to true, the transport chain is activated at application server startup.

Host: Specifies the host IP address to bind for transport. If the application server is on a local machine, the host name might be localhost.

Port: Specifies the port to bind for transport. The port number can be any port that currently is not in use on the system, might be localhost or the wildcard character * (an asterisk). The port number must be unique for each application server instance on a given machine

SSL Enabled: When set to true, users are notified if there is a channel that enables Secure Sockets Layer (SSL) in the listed chain. When SSL is enabled, all traffic going through this transport is encrypted and digitally secured.

Transport chain settings

This page lists the types of transport channels configured for the selected transport chain. A transport chain consists of one or more types of channels, each of which supports a different type of I/O protocol, such as TCP, HTTP, or DCS.

To view this administrative console page, click **Servers > Application servers >** *server_name* **> Ports**. Click on **View associated transports** for the port whose transport chains you want view and then click on the name of a specific chain.

Name:

Specifies the name of the selected transport chain.

You can edit this field to rename this transport chain. However, remember that the name must be unique within a WebSphere Application Server configuration.

Enabled: When checked, this transport chain is activated at application server startup.

Transport channels: Lists the transport channels configured for this transport chain and their configuration settings. To change a transport channel's configuration settings, click on the name of that transport channel.

HTTP tunnel transport channel settings

Use this page to view and configure an HTTP tunnel transport channels. Inbound connections sent through this channel are tunneled over HTTP, allowing intermediates to view this data as the body of an HTTP message instead of in its natural format. This type of channel is often used to circumvent firewalls with protocol restrictions.

To view this administrative console page, click **Servers > Application servers >***server_name* **> Ports** . Click on **View associated transports** for the port associated with the HTTP Tunnel transport channel whose settings you want to look at.

Transport channel name:

Specifies the name of the HTTP tunnel transport channel.

This name must be unique across all channels in a WebSphere Application Server environment. For example DCS and TCP transport channels cannot have the same name if they reside within the same system.

Discrimination weight:

Specifies the priority this channel has in relation to the other channels in this transport chain. This property is only used when port sharing is enabled and the channel chain includes multiple channels to which it might forward data. The channel in the chain with the lowest discrimination weight is the first one given the opportunity to look at incoming data and determine whether or not it owns that data.

Data type	Positive integer
Default	10

HTTP transport channel settings

Use this page to view and configure an HTTP transport channel. This type of transport channel handles HTTP requests from a remote client.

An HTTP transport channel parses HTTP requests and then finds an appropriate application channel to handle the request and send a response.

To view this administrative console page, click **Servers > Application servers >***server_name* **> Ports** . Click on **View associated transports** for the port associated with the HTTP transport channel whose settings you want to look at.

Transport channel name:

Specifies the name of the HTTP transport channel.

This name must be unique across all channels in a WebSphere Application Server environment. For example, an HTTP transport channel and a TCP transport channel cannot have the same name if they reside within the same system.

Discrimination weight:

Specifies the priority this channel has in relation to the other channels in this transport chain. This property is only used when port sharing is enabled and the channel chain includes multiple channels to which it might forward data. The channel in the chain with the lowest discrimination weight is the first one given the opportunity to look at incoming data and determine whether or not it owns that data.

Data type	Positive integer
Default	10

Maximum persistent requests:

Specifies the maximum number of persistent (keep-alive) requests that are allowed on a single HTTP connection. If a value of 0 (zero) is specified, only one request is allowed per connection. If a value of -1 is specified, an unlimited number of requests is allowed per connection.

Data type	Integer
Default	100

Use Keep-Alive: When selected, the HTTP transport channel, when sending an outgoing HTTP message, uses a persistent connection (keep-alive connection) instead of a connection that closes after one request or response exchange occurs.

Note: If a value other than 0 is specified for the `maximum persistent requests` property, the `Use Keep-Alive property` setting is ignored.

The default for this property is selected.

Read timeout:

Specifies the amount of time, in seconds, the HTTP transport channel waits for a read request to complete on a socket after the first read request occurs. The read being waited for could be an HTTP body (such as a POST) or part of the headers if they were not all read as part of the first read request on the socket.

Data type	Integer
Default	60 seconds

Write timeout:

Specifies the amount of time, in seconds, that the HTTP transport channel waits on a socket for each portion of response data to be transmitted. This timeout usually only occurs in situations where the writes are lagging behind new requests. This can occur when a client has a low data rate or the server's network interface card (NIC) is saturated with I/O.

Data type	Integer
Default	60 seconds

Persistent timeout:

Specifies the amount of time, in seconds, that the HTTP transport channel allows a socket to remain idle between requests.

Data type	Integer
Default	30 seconds

Enable NCSA access logging:

When selected, the HTTP transport channel performs NCSA access and error logging. Enabling NCSA access and error logging slows server performance.

To configure NCSA access and error logging, click **HTTP error and NCSA access logging** under **Related Items**. Even if HTTP error and NCSA access logging is configured, it is not enabled unless the Enable NCSA access logging property is selected.

The default value for the Enable NCSA access logging property is not selected.

TCP transport channel settings

Use this page to view and configure an TCP transport channels. This type of transport channel handles inbound TCP/IP requests from a remote client.

To view this administrative console page, click **Servers > Application servers >***server_name* **> Ports >** . Click on **View associated transports** for the port associated with the TCP transport channel whose settings you want to view.

Transport channel name:

Specifies the name of the TCP transport channel.

This name must be unique across all channels in a WebSphere Application Server environment. For example, a TCP transport channel and an HTTP transport channel cannot have the same name if they reside within the same system.

Port: Specifies the TCP/IP port this transport channel uses to establish connections between a client and an application server. The TCP transport channel binds to the hostnames and ports listed for the Port property. You can specify the wildcard * (an asterisk), for the hostname if you want this channel to listen to all hosts that are available on this system. However, before specifying the wildcard value, make sure this TCP transport channel does not have to bind to a specific hostname.

Thread pool: Select from the drop-down list of available thread pools the thread pool you want the TCP transport channel to use when dispatching work.

Maximum open connections:

Specifies the maximum number of connections that can be open at one time.

Data type	Integer between 1 and 20,000 inclusive
Default	20,000

Inactivity timeout: Specifies the amount of time, in seconds, that the TCP transport channel waits for a read or write request to complete on a socket.

Note: The value specified for this property might be overridden by the wait times established for channels above this channel. For example, the wait time established for an HTTP transport channel overrides the value specified for this property for every operation except the initial read on a new socket.

Data type	Integer
Default	60 seconds

Address exclude list:

Lists the IP addresses that are not allowed to make inbound connections. Use a comma to separate the IPv4 or IPv6 or both addresses to which you want to deny access on inbound TCP connection requests.

All four numeric values in an IPv4 address must be represented by a number or the wildcard character * (an asterisk).

Following are examples of valid IPv4 addresses that can be included in an **Address exclude list**:

```
*.1.255.0
254.*.*.9
1.*.*.*
```

All eight numeric values of an IPv6 address must be represented by a number or the wildcard character * (an asterisk). No shortened version of the IPv6 address should be used. Even though a shortened version is processed with no error given, it does not function correctly in this list. Each numeric entry should be a 1- 4 digit hexadecimal number.

Following are examples of valid IPv6 addresses that can be included in an **Address exclude list**:

```
0:*:*:0:007F:0:0001:0001
F:FF:FFF:FFFF:1:01:001:0001
1234:*:4321:*:9F9f:*:*:0000
```

Note: The **Address include list** and **Host name include list** are processed before the **Address exclude list** and the **Host name exclude list**. If all four lists are defined:

- An address that is defined on either inclusion list will be allowed access provided it is not included on either of the exclusion lists.
- If an address is included in both an inclusion list and in an exclusion list, it will not be allowed access.

Address include list:

Lists the IP addresses that are allowed to make inbound connections. Use a comma to separate the IPv4 or IPv6 or both addresses to which you want to grant access on inbound TCP connection requests.

All four numeric values in an IPv4 address must be represented by a number or the wildcard character * (an asterisk).

Following are examples of valid IP addresses that can be included in an **Address include list**:

```
 *.1.255.0
 254.*.*.9
 1.*.*.*
```

All eight numeric values of an IPv6 address must be represented by a number or the wildcard character * (an asterisk). No shortened version of the IPv6 address should be used. Even though a shortened version is processed with no error given, it does not function correctly in this list. Each numeric entry should be a 1- 4 digit hexadecimal number.

Following are examples of valid IPv6 addresses that can be included in an **Address include list**:

```
0:*:*:0:007F:0:0001:0001
F:FF:FFF:FFFF:1:01:001:0001
1234:*:4321:*:9F9f:*:*:0000
```

Note: The **Address include list** and **Host name include list** are processed before the **Address exclude list** and the **Host name exclude list**. If all four lists are defined:

- An address that is defined on either inclusion list will be allowed access provided it is not included on either of the exclusion lists.
- If an address is included in both an inclusion list and in an exclusion list, it will not be allowed access.

Host name exclude list:

List the host names that are not allowed to make connections. Use a comma to separate the URL addresses to which you want to deny access on inbound TCP connection requests.

A URL address can start with the wildcard character * (an asterisk) followed by a period; for example, `*.Rest.Of.Address`. If a period does not follow the wildcard character, the asterisk will be treated as a normal non-wildcard character. The wildcard character can not appear any where else in the address. For example, ibm.*.com is not a valid hostname.

Following are examples of valid URL addresses that can be included in a **Host name exclude list**:

```
*.ibm.com
www.ibm.com
*.com
```

Note: The **Address include list** and **Host name include list** are processed before the **Address exclude list** and the **Host name exclude list**. If all four lists are defined:

- An address that is defined on either inclusion list will be allowed access provided it is not included on either of the exclusion lists.
- If an address is included in both an inclusion list and in an exclusion list, it is not allowed access.

Host name include list:

Lists the host names that are allowed to make inbound connections. Use a comma to separate the URL addresses to which you want to grant access on inbound TCP connection requests.

A URL address can start with the wildcard character * (an asterisk) followed by a period; for example, `*.Rest.Of.Address`. If a period does not follow the wildcard character, the asterisk will be treated as a normal non-wildcard character. The wildcard character can not appear any where else in the address. For example, ibm.*.com is not a valid hostname.

Following are examples of valid URL addresses that can be included in a **Host name include list**:

```
*.ibm.com
www.ibm.com
*.com
```

Note: The **Address include list** and **Host name include list** are processed before the **Address exclude list** and the **Host name exclude list**. If all four lists are defined:

- An address that is defined on either inclusion list will be allowed access provided it is not included on either of the exclusion lists.
- If an address is included in both an inclusion list and in an exclusion list, it is not allowed access.

DCS transport channel settings

Use this page to view and configure an DCS transport channels. This type of transport channel handles inbound Distribution and Consistency Services (DCS) messages.

By default, two channel transport chains are defined for an application server that contains a DCS channel:

- The chain named DCS contains a TCP and a DCS channel.
- The chain named DCS-Secure contains a TCP, an SSL, and a DCS channel.

Both of these chains terminate in, or use the same TCP channel instance. This TCP channel is associated with the DCS_UNICAST_ADDRESS port and is not used in any other transport chains. One instance of an SSL channel is reserved for use in the DCS-Secure chain. It also is not used in any other transport chains.

To view this administrative console page, click **Servers > Application servers >***server_name* **> Ports >** . Click on **View associated transports** for the port associated with the DCS transport channel whose settings you want to look at.

Transport channel name:

Specifies the name of the DCS transport channel.

This name must be unique across all channels in a WebSphere Application Server environment. For example DCS and TCP transport channels cannot have the same name if they reside within the same system.

Discrimination weight:

Specifies the priority this channel has in relation to the other channels in this transport chain. This property is only used when port sharing is enabled and the channel chain includes multiple channels to which it might forward data. The channel in the chain with the lowest discrimination weight is the first one given the opportunity to look at incoming data and determine whether or not it owns that data.

The discrimination weight of the DCS channel in a DCS-Secure transport chain should always be less than the discrimination weight of the SSL channel that is in that chain. Other SSL channels in other chains might have different discrimination values.

Data type	Positive integer	
Default	1 for the DCS channel	2 for the SSL channel

Web container transport channel settings

Use this page to view and configure a Web container inbound channel transport. This type of channel transport handles inbound Web container requests from a remote client.

To view this administrative console page, click **Servers** > **Application servers** > *server_instance* > **Web container settings** > **Web container transport chains** > *transport_chain* > **Web Container Inbound Channel** .

Transport Channel Name:

This name must be unique across all channels in a WebSphere Application Server environment. This means that TCP transport channels and HTTP transport channels cannot have the same name if they reside within the same system.

Discrimination weight:

Specifies the priority that this transport chain has in relation to other transport chains if this transport channel is shared amongst several transport chains.

Write buffer size:

Specifies the amount of content in bytes to buffer unless the servlet explicitly calls flush/close on the response/writer output stream.

Data type	bytes
Default	9192 bytes

Custom services

A custom service provides the ability to plug into a WebSphere Application Server application server to define a hook point that runs when the server starts and shuts down.

A developer implements a custom service containing a class that implements a particular interface. The administrator configures the custom service in the administrative console, identifying the class created by the developer. When an application server starts, any custom services defined for the application server are loaded and the server runtime calls their initialize methods.

Developing custom services

The following restrictions apply to the WebSphere Application Server custom services implementation:

- The init and shutdown methods must return control to the runtime.
- No work is dispatched into the server instance until all custom service initialize methods return.
- The init and shutdown methods are called only once on each service, and once for each operating system process that makes up the server instance. File I/O is supported.
- Initialization of process level static data, without leaving the process, is supported.
- Only JDBC RMLT (resource manager local transaction) operations are supported. Every unit of work (UOW) must be completed before the methods return.
- Creation of threads is not supported.
- Creation of sockets and I/O, other than file I/O, is not supported. Running standard J2EE code (client code, servlets, enterprise beans) is not supported.
- The JTA interface is not available. This feature is available in J2EE server processes and distributed generic server processes only.
- While the runtime makes an effort to call shutdown, there is no guarantee that shutdown will be called prior to process termination.

Note that these restrictions apply to the shutdown and init methods equally. Some JNDI operations are available.

Develop a custom service class that implements the com.ibm.websphere.runtime.CustomService interface. The properties passed by the application server runtime to the initialize method can include one for an external file containing configuration information for the service (retrieved with externalConfigURLKey). In addition, the properties can contain any name-value pairs that are stored for the service, along with the other system administration configuration data for the service. The properties are passed to the initialize method of the service as a Properties object.

There is a shutdown method for the interface as well. Both methods of the interface declare that they may create an exception, although no specific exception subclass is defined. If an exception is created, the runtime logs it, disables the custom service, and proceeds with starting the server.

As mentioned above, your custom services class must implement the CustomService interface. In addition, your class must implement the initialize and shutdown methods. Suppose the name of the class that implements your custom service is *ServerInit*, your code would declare this class as shown below. The code below assumes that your custom services class needs a configuration file. It shows how to process the input parameter in order to get the configuration file. If your class does not require a configuration file, the code that processes configProperties is not needed.

```
public class ServerInit implements CustomService
{
/**
* The initialize method is called by the application server run-time when the
* server starts. The Properties object passed to this method must contain all
* configuration information necessary for this service to initialize properly.
*
* @param configProperties java.util.Properties
*/
    static final java.lang.String externalConfigURLKey =
       "com.ibm.websphere.runtime.CustomService.externalConfigURLKey";

    static String ConfigFileName="";

    public void initialize(java.util.Properties configProperties) throws Exception
    {
        if (configProperties.getProperty(externalConfigURLKey) != null)
        {
           ConfigFileName = configProperties.getProperty(externalConfigURLKey);
        }

       // Implement rest of initialize method
    }
/**
* The shutdown method is called by the application server run-time when the
* server begins its shutdown processing.
*
* @param configProperties java.util.Properties
*/
    public void shutdown() throws Exception
    {
        // Implement shutdown method
    }
```

Custom service collection

Use this page to view a list of services available to the application server and to see whether the services are enabled. A custom service provides the ability to plug into a WebSphere application server and define code that runs when the server starts or shuts down.

To view this administrative console page, click **Servers > Application servers >***server_name*. Then, under Server Infrastructure, click **Administration > Custom Services**.

External Configuration URL:

Specifies the URL for a custom service configuration file.

If your custom services class requires a configuration file, the value provides a fully-qualified path name to that configuration file. This file name is passed into your custom service class.

Classname:

Specifies the class name of the service implementation. This class must implement the Custom Service interface.

Display Name:

Specifies the name of the service.

Enable service at server startup:

Specifies whether the server attempts to start and initialize the service when its containing process (the server) starts. By default, the service is not enabled when its containing process starts.

Custom service settings:

Use this page to configure a service that runs in an application server.

To view this administrative console page, click **Servers > Application servers >***server_name*. Then, under Server Infrastructure, click **Administration > Custom services >***custom_service_name*.

Enable service at server startup:

Specifies whether the server attempts to start and initialize the service when its containing process (the server) starts. By default, the service is not enabled when its containing process starts.

Data type	Boolean
Default	false

External Configuration URL:

Specifies the URL for a custom service configuration file.

If your custom services class requires a configuration file, specify the fully-qualified path name to that configuration file for the value. This file name is passed into your custom service class.

Data type	String
Units	URL

Classname:

Specifies the class name of the service implementation. This class must implement the Custom Service interface.

Data type	String
Units	Java class name

Display Name:

Specifies the name of the service.

Data type	String

Description:

Describes the custom service.

Data type	String

Classpath:

Specifies the class path used to locate the classes and JAR files for this service.

Data type	String
Units	Class path

Process definition

A process definition specifies the run-time characteristics of an application server process.

A process definition can include characteristics such as JVM settings, standard in, error and output paths, and the user ID and password under which a server runs.

Defining application server processes

To enhance the operation of an application server, you can define command-line information for starting or initializing an application server process. Such settings define run-time properties such as the program to run, arguments to run the program, and the working directory.

1. Go to the settings page for a process definition in the administrative console. Click **Servers > Application Servers** in the console navigation tree, click on an application server name and then **Java and Process Management > Process Definition**. Note that you can also define application server processes using the wsadmin tool. For more information, see the *Administering applications and their environment* PDF.
2. On the settings page for a process definition, specify the name of the executable to run, any arguments to pass when the process starts running, and the working directory in which the process will run. Then click **OK**.
3. Specify process execution statements for starting or initializing a UNIX process.
4. Specify monitoring policies to track the performance of a process.
5. Specify process logs to which standard out and standard error streams write. Complete this step if you do not want to use the default file names.
6. Specify name-value pairs for properties needed by the process definition.
7. Stop the application server and then restart the server.
8. Check the application server to ensure that the process definition runs and operates as intended.

Process definition settings

Use this page to view or change settings for a process definition. For WebSphere Application Server, this page provides command-line information for starting or initializing a process.

To view this administrative console page, click **Servers > Application Servers >***server_name*. Then under Server Infrastructure click **Java Process Management > Process Definition**.

Related concepts

"Process definition"
A process definition specifies the run-time characteristics of an application server process.

Related tasks

"Defining application server processes" on page 145

Related reference

"Administrative console page features" on page 178
This topic provides information about the basic elements of an administrative console page, such as the various tabs.

"Java virtual machine settings" on page 155
Use this page to view and change the Java virtual machine (JVM) configuration for the application server's process.

"Custom property collection" on page 118
Use this page to view and manage arbitrary name-value pairs of data, where the name is a property key and the value is a string value that can be used to set internal system configuration properties.

Executable Name:

This command applies to base WebSphere Application Server only. It specifies the executable name that is invoked to start the process.

Data type	String

Executable Arguments:

This command applies to base WebSphere Application Server only. It specifies the arguments that are passed to the executable when starting the process.

For example, the executable target program might expect three arguments: *arg1 arg2 arg3*.

Data type	String
Units	Java command-line arguments

Working Directory:

Specifies the file system directory that the process uses as its current working directory.

The process uses this directory to determine the locations of input and output files with relative path names.

Passivated enterprise beans are placed in the current working directory of the application server on which the beans are running. Make sure the working directory is a known directory under the root directory of the WebSphere Application Server product.

Data type	String

Process execution settings:

Use this page to view or change the process execution settings for a server process that applies to either an application server, a node agent or a deployment manager.

To view this administrative console page for an application server, click **Servers > Application Servers >***server_name*. Then, under Server Infrastructure , click **Java and Process Management > Process Execution**.

To view this administrative console page for a node agent, click **System Administration > Node agents >***node_agent_name*. Then, under Server Infrastructure , click **Java and Process Management > Process Definition > Process Execution**.

To view this administrative console page for a deployment manager, click **System Administration > Deployment manager**. Then, under Server Infrastructure , click **Java and Process Management > Process Definition > Process Execution**.

Process Priority:

Specifies the operating system priority for the process. The administrative process that launches the server must have root operating system authority in order to honor this setting.

Data type	Integer
Default	20 for WebSphere Application Server on all operating systems.

UMASK:

Specifies the user mask under which the process runs (the file-mode permission mask).

Data type	Integer

Run As User:

Specifies the user that the process runs as.

Data type	String

Run As Group:

Specifies the group that the process is a member of and runs as.

On OS/400, the Run As Group setting is ignored.

Data type	String

Run In Process Group:

Specifies a specific process group for the process. This process group is useful for such things as processor partitioning. A system administrator can assign a process group to run on, for example, 6 of 12 processors. The default (0) is not to assign the process to any specific group.

On OS/400, the Run In Process Group setting is ignored.

Data type	Integer
Default	0

Process logs settings:

Use this page to view or change settings for specifying the files to which standard out and standard error streams write.

To view this administrative console page, click **Servers > Application Servers >***server_name*. Then, under Troubleshooting, click **Logging and Tracing > Process Logs**.

Stdout File Name:

Specifies the file to which the standard output stream is directed. The file name can include a symbolic path name defined in the variable entries.

Use the field on the configuration tab to specify the file name. Use the field on the Runtime tab to select a file for viewing. View the file by clicking **View**.

Direct server output to the administrative console or to the process that launched the server, by either deleting the file name or specifying `console` on the configuration tab.

Data type	String
Units	File path name

Stderr File Name:

Specifies the file to which the standard error stream is directed. The file name can include a symbolic path name defined in the variable entries.

Use the field on the configuration tab to specify the file name. Use the field on the runtime tab to select a file for viewing. View the file by clicking **View**.

Data type	String
Units	File path name

Monitoring policy settings:

Use this page to view or change settings that control how the node agent monitors and restarts a process.

To view this administrative console page, click **Servers > Application Servers >***server_name*. Then, under Server Infrastructure, click **Java and Process Management > Process Definition > Monitoring Policy**.

Maximum Startup Attempts:

Specifies the maximum number of times to attempt to start the application server before giving up.

Data type	Integer

Ping Interval:

Specifies the frequency of communication attempts between the parent process, such as the node agent, and the process it has spawned, such as an application server. Adjust this value based on your requirements for restarting failed servers. Decreasing the value detects failures sooner; increasing the value reduces the frequency of pings, reducing system overhead.

Data type	Integer
Range	Set the value greater than or equal to 0 (zero) and less than 2147483.

Ping Timeout:

When a parent process is spawning a child process, such as when a process manager spawns a server, the parent process pings the child process to see whether the child was spawned successfully. This value specifies the number of seconds that the parent process should wait (after pinging the child process) before assuming that the child process failed.

Data type	Integer
Units	Seconds
Range	Set the value greater than or equal to 0 (zero) and less than 2147483.

Automatic Restart:

Specifies whether the process should restart automatically if it fails. The default is to restart the process automatically.

Data type	Boolean
Default	true

Node Restart State:

Specifies the desired state for the process after the node completely shuts down and restarts.

Data type	String
Default	STOPPED
Range	Valid values are STOPPED, RUNNING, or PREVIOUS. If you want the process to return to its current state after the node restarts, use PREVIOUS.

Automatically restarting server processes

There are several server processes related to WebSphere Application Server products that the operating system can monitor and automatically restart when the server processes stop abnormally. This task describes how to set up these *monitored* processes.

To set up this function on a Linux or UNIX-based operating system, you must have root authority to edit the inittab file.

On a Windows operating system, you must belong to the Administrator group and have the following advanced user rights:

- Act as part of the operating system
- Log on as a service

The Installation wizard grants you the user rights if your user ID is part of the administrator group. If you are running on a Microsoft Windows 2000 Operating System, the Installation wizard displays a message that states that although the advanced user rights are now effective, they do not display as effective until the next time you log on to the Windows machine.

You can also add the advanced user rights manually if you are performing a silent installation on a Windows platform. For example, to grant the user rights to your administrator group user ID on a Windows 2000 Server platform, perform the following procedure:

1. Click **Administrative Tools** in the Control Panel.
2. Click **Local Security Policy**.
3. Click **Local Policies**.
4. Click **User Rights Assignments**.
5. Right click **Act as part of the operating system**.
6. Click **Security**.
7. Click **Add**.
8. Click your user ID.

9. Click **Add**.
10. Click **OK**.
11. Click **OK**.
12. Right click **Log on as a service**.
13. Click **Security**.
14. Click **Add**.
15. Click **OK**.
16. Click **OK**.
17. Reboot your machine to make the settings effective.

Consult your Windows help system for more information.

You can use this function to automatically restart Express servers. You can restart the ***server1*** process, for example.

You must manually create a shell script that automatically starts any of the processes previously mentioned, on a Linux and UNIX-based operating system. Each Windows service or UNIX shell script controls a single process, such as a stand-alone WebSphere Application Server instance. Multiple stand-alone Application Server processes require multiple Windows service or UNIX scripts, which you can define.

If you do not install the WebSphere Application Server base product as a Windows service during installation, you can use the **WASService** command in the *install_root*/bin directory to do so at a later time. You can use the tool to add any WebSphere Application Server process as a Windows service. The operating system can then monitor each server process and restart the process if it stops.

1. **Use the installation wizard** to set up a Windows service to automatically monitor and restart processes related to the WebSphere Application Server product.
   - Perform the following procedure from the installation wizard to select services that the installation wizard can set up:
     a. Click **Run WebSphere Application Server as a service**.

        If you select this option, the installation wizard creates the following service during the installation:

        IBMWAS6Service - *node_name*

        The **IBMWAS6Service -***node_name* service controls the *node_name* process.

        After you complete and verify the installation, use the Windows Services panel to change the **IBMWAS6Service -***node_name* service to an automatic startup type.
        1) Right click **IBMWAS6Service -***node_name* and click **Properties**.
        2) Click **Automatic** from the **Startup type** list box and click **OK**.

     b. Click **Run IBM HTTP Server as a service**.

        Select this option on the machine where you are installing the IBM HTTP Server.

        If you select this option, the installation wizard creates the following services during the installation:
        – **IBM HTTP Server 2.0.*x***
        – **IBM HTTP Administration 2.0.*x***

        The installation wizard defines the startup type of these services as **automatic**. It is not necessary for you to change the type from manual to automatic.

     c. Enter your user ID and password and click **Next**.

     In a coexistence environment, you can change the default service names to make them unique. In a same version coexistence scenario for IBM HTTP Server 2.0.x on a Windows platform, you cannot use the default service names created by the installer because they are common.

     To work around this problem:

a. Install the first copy of IBM HTTP Server, either by itself or with WebSphere Application Server and select to install the services.
b. Customize the service names for the first install by running the following commands from the first install location:

```
apache -k install -n "IHS 2.0(1)"
apache -k install -f conf\admin.conf -n "IHS 2.0 Administration (1)"
```

c. Edit the AdminAlias directive in the *installLocation 1*`\conf\admin.conf` file to point to the new service name, such as **IHS 2.0(1)**.
d. Remove the default service names installed by the first install by running the following commands:

```
apache -k uninstall -n "IBM HTTP Server 2.0"
apache -k uninstall -n "IBM HTTP Administration 2.0"
```

e. Install the second copy of IBM HTTP Server, either by itself or with WebSphere Application Server. The default service names correspond to the second install.

Note: Customized service names must be unique on your system.

2. **After installing**, you can use the WASService.exe utility in the *install_root*`\bin` directory to manually define a Windows service for another installation instance or for another configuration instance of the server1 process.

You can use the **net start** and **net stop** commands to control the IBM HTTP Server services on a Windows system. For more information about these commands, see the Windows help file. Access these commands from the Start menu, clicking **Start > Programs > IBM HTTP Server**.

You can also use the **Start the Server** and **Stop the Server** commands to control the IBM WebSphere Application Server process on a Windows system. Access these commands from the Start menu, clicking **Start > Programs > IBM WebSphere > Application Server V6**.

Processes started by a **startServer**command are not running as monitored processes, regardless of how you have configured them.

For example, you can configure a server1 process as a monitored process. However, if you start the server1 process using the **startServer** command, the operating system does not monitor or restart the server1 process because the operating system did not originally start the process as a monitored process.

Return to Defining application server processes to continue.

WASService command:

The **WASService** command line tool lets you create a Windows service for any WebSphere Application Server Java process.

You can create Windows services for WebSphere Application Server Java processes. Potential Windows services include the following server processes:

- The default server1 process on an application server node
- Application server processes that you create on an application server node

When the installation wizard creates a Windows service, the uninstaller program can remove the Windows service. If you use the **WASService** command to create a service yourself, it is your responsibility to remove the service when it is no longer valid. The uninstaller program does not remove Windows services that you create with the **WASService** command.

Location of the command file: The `WASService.exe` command file is located in the *install_root*`\bin` directory.

Command syntax:

WASService.exe command syntax for starting an existing service

The command syntax is as follows:

```
WASService.exe [-start] "service_name" [optional startServer.bat parameters]
```

WASService.exe command syntax for creating a service or updating an existing service

The command syntax is as follows:

```
WASService.exe -add "service_name"
                 -serverName server
                 -profilePath server_profile_directory

                [-wasHome  install_root]
             [-configRoot configuration_repository_directory]
               [-startArgs additional_start_arguments]
               [-stopArgs additional_stop_arguments]
               [-userid user_id -password password]
               [-logFile service_log_file]
               [-logRoot server_log_directory]
               [-restart true | -restart false]
               [-startType automatic | manual | disabled]
```

WASService.exe command syntax for deleting a service

The command syntax is as follows:

```
WASService.exe -remove "service_name"
```

WASService.exe command syntax for stopping a running service

The command syntax is as follows:

```
WASService.exe -stop "service_name" [optional stopServer.bat parameters]
```

WASService.exe command syntax for retrieving service status

The command syntax is as follows:

```
WASService.exe -status "service_name"
```

WASService.exe command syntax for encoding parameters

The command syntax is as follows:

```
WASService.exe -encodeParams "service_name"
```

Parameters: Supported arguments include:

-add *"service_name"*
Creates a service named *service_name* or updates an existing Windows service. The syntax is the same for both cases.

-configRoot *configuration_repository_directory*
Optional parameter that identifies the configuration directory of the installation root directory of a WebSphere Application Server product.

-encodeParams *service_name*
Optional parameter that forces the service to encode the -startArgs and -stopArgs so that the arguments cannot be determined by editing the registry. Use the parameter when creating a service with the -add parameter by adding -encodeParams to the command line with no arguments. Or encode the parameters of an existing service:

```
WASService -encodeParams service_name
```

-logFile *service_log_file*
Optional parameter that identifies a log file that the **WASService** command uses to record its activity.

-logRoot *server_log_directory*
Required parameter that identifies the server log directory for the profile. The **WASService** command looks for a file named `server_name.pid` to determine if the server is running.

-profilePath *server_profile_directory*
Specifies the directory path of the profile that defines the server process.

-remove *service_name*
Deletes the specified service.

-restart true | false
Restarts the existing service automatically if the service fails when set to true.

-serverName *Server_name*
Identifies the server that the service controls.

-start *"service_name"* **[optional startServer.bat parameters]**
Starts the existing service.

-startArgs *additional_start_arguments*
Optional parameter that identifies additional parameters.

-startType automatic | manual | disabled
Defines the startup type of the new service. An automatic startup type starts automatically when the system starts or when the service is called for the first time. You must start a manual service before the operating system can load it and make it available. You cannot start a disabled service before changing the startup type.

-status *service_name*
Returns the current status of the service, which includes whether the service is running or stopped.

-stop *service_name* **[optional stopServer.bat parameters]**
Stops the specified service.

-stopArgs *additional_stop_arguments*
Optional parameter that identifies additional parameters.

-userid *user_ID* **-password** *password*
Optional parameters that identify a privileged user ID and password that the Windows service will run as.

-wasHome *install_root*
Optional parameter that identifies the installation root directory of the WebSphere Application Server product.

Default names for Windows services that are created by the wizard: The name of the Windows service that is created by the Installation wizard is `IBM WebSphere Application Server V6 - DefaultNode`.

Viewing the Windows services panel: To view Windows services, open the Control panel and click **Administrative Tools > Services**. Select a service to view information about it. Right click the service and click **Properties**. Four tabs provide information and functionality. For example, select the **Setup type** field on the **General** tab to change the setup type.

Examples:
Creating an Application Server service

This example creates a service called *IBM WebSphere Application Server V6 - server2* that starts an Application Server process:

```
WASService -add server2
           -servername server2
           -profilePath "C:\Program Files\IBM\WebSphere\AppServer\
                     profiles\CustomProfile"
           -wasHome "C:\Program Files\IBM\WebSphere\AppServer"
           -logfile "C:\Program Files\IBM\WebSphere\AppServer\
                        profiles\CustomProfile\logs\startNode.log"
           -logRoot "C:\Program Files\IBM\WebSphere\AppServer\
                        profiles\CustomProfile\logs"
           -restart true
```

After entering the command, messages that are similar to those in the following example display in the command window:

```
Adding Service: server2
        Config Root: C:\Program Files\IBM\WebSphere\AppServer\
                        profiles\CustomProfile\config
        Server Name: server2
        Profile Path: C:\Program Files\IBM\WebSphere\AppServer\
                        profiles\CustomProfile
        Was Home: C:\Program Files\IBM\WebSphere\AppServer\
        Start Args:
        Restart: 1
IBM WebSphere Application Server V6 - server2 service successfully added.
```

Updating an existing Application Server service

This example updates an existing service called IBM WebSphere Application Server V6 - server2 with additional stop arguments, username and password. The user name and password are required by the **stopServer** command to stop the application server with security enabled.

```
WASService -add server2
           -servername server2
           -profilePath "C:\Program Files\IBM\WebSphere\AppServer\
                     profiles\CustomProfile"
           -stopArgs "-username user_name  -password password"
           -encodeParams server2
```

Starting and stopping a server process after creating a Windows service: If you issue the **startServer server1** command or the **stopServer server1** after creating a Windows service for server1, a message that is similar to the following example displays:

```
Because server1 is registered to run as a Windows Service, the
request to start this server will be completed by starting the
associated Windows Service.
```

Stopping a server after enabling security

If you enable security while a Windows service is running, you cannot stop the server from the command line, even when using the username and password parameters on the **stopServer** command. A message similar to the following example is displayed:

```
Could not stop the IBM WebSphere Application Server V6 -
server_name service on Local Computer. The service
did not return an error. This could be an internal Windows
error or an internal service error. If the problem persists,
contact your system administrator.
```

The problem is due to the service control of the process. You must change the service to use the proper stop-server arguments for a secure server.

Use the -stopArgs parameter and the -encodeParams parameter to update the service as described in the ″Updating an existing application server service″ example.

Java virtual machines (JVMs)

The Java virtual machine (JVM) is an interpretive computing engine responsible for running the byte codes in a compiled Java program. The JVM translates the Java byte codes into the native instructions of the host machine. The application server, being a Java process, requires a JVM in order to run, and to support the Java applications running on it. JVM settings are part of an application server configuration.

Using the JVM

As part of configuring an application server, you might define settings that enhance your system's use of the Java virtual machine (JVM).

To view and change the JVM configuration for an application server's process, use the Java Virtual Machine page of the administrative console or use the wsadmin tool to change the configuration through scripting.

1. Access the Java Virtual Machine page.
   a. Click **Servers > Application Servers** in the console navigation tree.
   b. On the Application Server page, click on the name of the server whose JVM settings you want to configure.
   c. On the settings page for the selected application server, click **Java and Process Management > Process Definition**.
   d. On the Process Definition page, click **Java Virtual Machine**.
2. On the Java Virtual Machine page, specify values for the JVM settings as needed and click **OK**.
3. Click **Save** on the console task bar.
4. Restart the application server.

″"Configuring application servers for UTF-8 encoding" on page 112″ provides an example that involves specifying a value for the **Generic JVM Arguments** property on the Java Virtual Machine page to enable UTF-8 encoding on an application server. Enabling UTF-8 allows multiple language encoding support to be used in the administrative console.

″"Configuring JVM sendRedirect calls to use context root" on page 159″ provides an example that involves defining a property for the JVM.

Java virtual machine settings

Use this page to view and change the Java virtual machine (JVM) configuration for the application server's process.

To view this administrative console page, click **Servers > Application Servers >***server_name* **> Process Definition > Java Virtual Machine**.

Classpath:

Specifies the standard class path in which the Java virtual machine code looks for classes.

Enter each classpath entry into a table row. You do not need to add the colon or semicolon at the end of each entry.

Data type	String
Units	Class path

Boot Classpath:

Specifies bootstrap classes and resources for JVM code. This option is only available for JVM instructions that support bootstrap classes and resources. You can separate multiple paths by a colon (:) or semi-colon (;), depending on operating system of the node.

Data type	String

Verbose Class Loading:

Specifies whether to use verbose debug output for class loading. The default is not to enable verbose class loading.

Data type	Boolean
Default	false

Verbose Garbage Collection:

Specifies whether to use verbose debug output for garbage collection. The default is not to enable verbose garbage collection.

Data type	Boolean
Default	false

Verbose JNI:

Specifies whether to use verbose debug output for native method invocation. The default is not to enable verbose Java Native Interface (JNI) activity.

Data type	Boolean
Default	false

Initial Heap Size:

Specifies the initial heap size available to the JVM code, in megabytes.

Increasing the minimum heap size can improve startup. The number of garbage collection occurrences are reduced and a 10% gain in performance is realized.

Increasing the size of the Java heap improves throughput until the heap no longer resides in physical memory, in general. After the heap begins swapping to disk, Java performance suffers drastically.

Data type	Integer
Default	The default is 50.

Maximum Heap Size:

Specifies the maximum heap size available to the JVM code, in megabytes.

Increasing the heap size can improve startup. By increasing heap size, you can reduce the number of garbage collection occurrences with a 10% gain in performance.

Increasing the size of the Java heap improves throughput until the heap no longer resides in physical memory, in general. After the heap begins swapping to disk, Java performance suffers drastically. Set the maximum heap size low enough to contain the heap within physical memory.

Data type	Integer
Default	0 for iSeries, 256 for all other platforms. Keep the value low enough to avoid paging or swapping-out-memory-to-disk.

Run HProf:

This setting applies to base WebSphere Application Server only. It specifies whether to use HProf profiler support. To use another profiler, specify the custom profiler settings using the HProf Arguments setting. The default is not to enable HProf profiler support.

If you set the Run HProf property to true, then you must specify command-line profiler arguments as values for the HProf Arguments property.

Data type	Boolean
Default	false

HProf Arguments:

This setting applies to base WebSphere Application Server only. It specifies command-line profiler arguments to pass to the JVM code that starts the application server process. You can specify arguments when HProf profiler support is enabled.

HProf arguments are only required if the Run HProf property is set to true.

Data type	String

Debug Mode:

Specifies whether to run the JVM in debug mode. The default is not to enable debug mode support.

If you set the Debug Mode property to true, then you must specify command-line debug arguments as values for the Debug Arguments property.

Data type	Boolean
Default	false

Debug Arguments:

Specifies command-line debug arguments to pass to the JVM code that starts the application server process. You can specify arguments when Debug Mode is enabled.

Debug arguments are only required if the Debug Mode property is set to true. If you enable debugging on multiple application servers on the same server, make sure that the servers are using different `address` arguments, which define the port for debugging. For example, if you enable debugging on two servers and leave the default debug port for each server as `address=7777`, the servers could fail to start properly.

Data type	String
Units	Java command-line arguments

Generic JVM Arguments:

Specifies command line arguments to pass to the Java virtual machine code that starts the application server process.

The following are optional command line arguments that you can use by entering them into the **General JVM Arguments** field. If you enter more than one argument, separate each argument by a space.

Note: If the argument says it is for the IBM Developer Kit only, you cannot use the argument with another JVM, such as the Sun JDK or the HP JDK.

- **-Xquickstart:** You can use **-Xquickstart** for initial compilation at a lower optimization level than in default mode. Later, depending on sampling results, you can recompile to the level of the initial compile in default mode. Use **-Xquickstart** for applications where early moderate speed is more important than long run throughput. In some debug scenarios, test harnesses and short-running tools, you can improve startup time between 15-20%.

  The **-Xquickstart** option is not supported on OS/400.
- **-Xverify:none:** When using this value, the class verification stage is skipped during class loading . By using **-Xverify:none** with the just in time (JIT) compiler enabled, startup time is improved by 10-15%.

  The **-Xverify:none** option is not supported on OS/400.
- **-Xnoclassgc:** You can use this value to disable class garbage collection, which leads to more class reuse and slightly improved performance. The trade-off is that you won't be collecting the resources owned by these classes. You can monitor garbage collection using the verbose:gc configuration setting, which will output class garbage collection statistics. Examining these statistics will help you understand the trade-off between the reclaimed resources and the amount of garbage collection required to reclaim the resources. However, if the same set of classes are garbage collected repeatedly in your workload, you should disable garbage collection. Class garbage collection is enabled by default.
- **-Xgcthreads:** You can use several garbage collection threads at one time, also known as *parallel garbage collection*. When entering this value in the **Generic JVM Arguments** field, also enter the number of processors that your machine has, for example, `-Xgcthreads=`*number_of_processors*. On a node with *n* processors, the default number of threads is *n*. You should use parallel garbage collection if your machine has more than one processor. This argument is valid only for the IBM Developer Kit.

  The **-Xgcthreads** option is not supported on OS/400.
- **-Xnocompactgc:** This value disables heap compaction which is the most expensive garbage collection operation. Avoid compaction in the IBM Developer Kit. If you disable heap compaction, you eliminate all associated overhead.
- **-Xinitsh:** You can use this value to set the initial heap size where class objects are stored. The method definitions and static fields are also stored with the class objects. Although the system heap size has no upper bound, set the initial size so that you do not incur the cost of expanding the system heap size, which involves calls to the operating system memory manager. You can compute a good initial system heap size by knowing the number of classes loaded in the WebSphere Application Server product, which is about 8,000 classes, and their average size. Having knowledge of the applications helps you include them in the calculation. You can use this argument only with the IBM Developer Kit.
- **-Xgpolicy:** You can use this value to set the garbage collection policy. If the garbage collection policy (gcpolicy) is set to `optavgpause`, concurrent marking is used to track application threads starting from the stack before the heap becomes full. The garbage collector pauses become uniform and long pauses are not apparent. The trade-off is reduced throughput because threads might have to do extra work. The default, recommended value is `optthruput`. Enter the value as `-Xgcpolicy:[optthruput|optavgpause]`. You can use this argument only with the IBM Developer Kit.
- **-XX:** The Sun-based Java Development Kit (JDK) Version 1.4.2 has generation garbage collection, which allows separate memory pools to contain objects with different ages. The garbage collection cycle collects the objects independently from one another depending on age. With additional parameters, you can set the size of the memory pools individually. To achieve better performance, set the size of the pool containing short lived objects so that objects in the pool do not live through more then one garbage collection cycle. The size of new generation pool is determined by the `NewSize` and `MaxNewSize` parameters. Objects that survive the first garbage collection cycle are transferred to another pool. The size of the survivor pool is determined by parameter `SurvivorRatio`. If garbage collection becomes a bottleneck, you can try customizing the generation pool settings. To monitor garbage collection statistics,

use the object statistics in Tivoli Performance Viewer or the verbose:gc configuration setting. Enter the following values: `-XX:NewSize (lower bound)` , `-XX:MaxNewSize (upper bound)`, and `-XX:SurvivorRatio=NewRatioSize`. The default values are:`NewSize=2m MaxNewSize=32m SurvivorRatio=2` However, if you have a JVM with more than 1 GB heap size, you should use the values: `-XX:newSize=640m -XX:MaxNewSize=640m -XX:SurvivorRatio=16`, or set 50 to 60% of total heap size to a new generation pool.

  The **-XX** option is not supported on OS/400.
- **-Xminf:** You can use this value to specify the minimum free heap size percentage. The heap grows if the free space is below the specified amount. In reset enabled mode, this option specifies the minimum percentage of free space for the middleware and transient heaps. This is a floating point number, 0 through 1. The default is .3 (30%).
- **-server | -client:** Java HotSpot Technology in the Sun-based Java Development Kit (JDK) Version 1.4.2 introduces an adaptive JVM containing algorithms for optimizing byte code execution over time. The JVM runs in two modes, **-server** and **-client**. If you use the default **-client** mode, there will be a faster startup time and a smaller memory footprint, but lower extended performance. You can enhance performance by using **-server** mode if a sufficient amount of time is allowed for the HotSpot JVM to warm up by performing continuous execution of byte code. In most cases, use **-server** mode, which produces more efficient run-time execution over extended periods. You can monitor the process size and the server startup time to check the difference between **-client** and**-server**.

  The **-server | -client** option is not supported on OS/400.

Data type	String
Units	Java command line arguments

Executable JAR File Name:

Specifies a full path name for an executable JAR file that the JVM code uses.

Data type	String
Units	Path name

Disable JIT:

Specifies whether to disable the just in time (JIT) compiler option of the JVM code.

If you disable the JIT compiler, throughput decreases noticeably. Therefore, for performance reasons, keep JIT enabled.

Data type	Boolean
Default	false (JIT enabled)
Recommended	JIT enabled

Operating System Name:

Specifies JVM settings for a given operating system. When started, the process uses the JVM settings for the operating system of the server.

Data type	String

Configuring JVM sendRedirect calls to use context root

If the com.ibm.websphere.sendredirect.compatibility property is not set and your application servlet code has statements such as *sendRedirect("/home.html")*, your Web browser might display messages such as *Error 404: No target servlet configured for uri: /home.html*. To instruct the server not to use the Web server's document root and to use instead the Web application's context root for sendRedirect() calls, configure the JVM by setting the com.ibm.websphere.sendredirect.compatibility property to a `true` or `false` value.

1. Access the settings page for a property of the JVM.
   a. Click **Servers > Application Servers** in the console navigation tree.
   b. On the Application Server page, click on the name of the server whose JVM settings you want to configure.
   c. On the settings page for the selected application server, under Server Infrastructure, click **Java and Process Management > Process Definition**.
   d. On the Process Definition page, click **Java Virtual Machine**.
   e. On the Java Virtual Machine page, click **Custom Properties**.
   f. On the Custom Properties page, click **New**.
2. On the settings page for a property, specify a name of `com.ibm.websphere.sendredirect.compatibility` and either `true` or `false` for the value, then click **OK**.
3. Click **Save** on the console task bar.
4. Stop the application server and then restart the application server.

Setting custom JVM properties

In the WebSphere Application Server administrative console, you can change the values of the following custom JVM properties:

com.ibm.websphere.network.useMultiHome

For a distributed platform:

Set this property in a multihomed environment where WebSphere Application Server is restricted to listen only on a specific IP address for Discovery and SOAP messages.

The settings for the **com.ibm.websphere.network.useMultiHome** property are as follows:

- Setting this property to `false` specifies that WebSphere Application Server will listen on all IP addresses on the host for Discovery and SOAP messages.
- Setting this property to `true` specifies that WebSphere Application Server will only listen on the configured host name for Discovery and SOAP messages. If you set this property to `true`, you should have a host name configured on WebSphere Application Server that resolves to a specific IP address.
- Setting this property to `null` specifies that WebSphere Application Server will only listen on the default IP address only.

If you cannot contact the server, check the setting for **com.ibm.websphere.network.useMultihome** to ensure it is correct.

You can change the value through the administrative console. Modify the defaults by setting the value for the server, deployment manager, and node agent. In order for these changes to take place, you must restart the server.

Steps for this task

1. To set this property, connect to the administrative console and navigate to the indicated page.

Application server	**Servers > Application Servers >***server1***> Process Definition >***Control* **> Java Virtual Machine > Custom Properties**
Deployment manager	**System Administration > Deployment Manager > Process definition >** *Control* **> Java Virtual Machine > Custom Properties**

Node agent	**System Administration >Node Agent >** *nodeagent* **> Process definition >***Control* **> Java Virtual Machine > Custom Properties**

2. If the **com.ibm.websphere.network.useMultiHome** property is not present in the list, create a new property name and indicate its value.
3. Restart the server.

com.ibm.websphere.deletejspclasses

Deletes JavaServer Pages classes for all applications after those applications have been deleted or updated. By default, the value of this property is `true`.

Steps for this task

1. Connect to the administrative console and navigate to the Java Virtual Machine Custom Properties panel.

   For a distributed platform:

Base configuration	**Servers > Application Servers >***server1*. Then, under Server Infrastructure, click **Java and Process Management > Process definition > Java Virtual Machine > Custom Properties**
ND configuration	**System Administration > Node Agents >***nodeagent*. Then, under Server Infrastructure, click **Java and Process Management > Process definition > Java Virtual Machine > Custom Properties**

2. If the **com.ibm.websphere.deletejspclasses** property is not present in the list, create a new property name.
3. Enter the name and value.

com.ibm.websphere.deletejspclasses.delete

Deletes JavaServer Pages classes for all applications after those applications have been deleted, but not after they have been updated. By default, the value of this property is `true`.

Steps for this task

1. Connect to the administrative console and navigate to the Java Virtual Machine Custom Properties panel.

   For a distributed platform:

Base configuration	**Servers > Application Servers >***server1* **> Process definition > Java Virtual Machine > Custom Properties**
ND configuration	**System Administration > Node Agents >***nodeagent*. Then, under Server Infrastructure, click **Java and Process Management > Process definition > Java Virtual Machine > Custom Properties**

2. If the **com.ibm.websphere.deletejspclasses.delete** property is not present in the list, create a new property name.
3. Enter the name and value.

com.ibm.websphere.deletejspclasses.update

Deletes JavaServer Pages classes for all applications after those applications have been updated, but not after they have been deleted. By default, the value of this property is `true`.

Steps for this task

1. Connect to the administrative console and navigate to the Java Virtual Machine Custom Properties panel.

For a distributed platform:

Base configuration	**Servers > Application Servers >***server1*. Then, under Server Infrastructure, click **Java and Process Management > Process definition > Java Virtual Machine > Custom Properties**
ND configuration	**System Administration > Node Agents >***nodeagent*. Then, under Server Infrastructure, click **Java and Process Management > Process definition > Java Virtual Machine> Custom Properties**

2. If the **com.ibm.websphere.deletejspclasses.update** property is not present in the list, create a new property name.
3. Enter the name and value.

Tuning Java virtual machines

The application server, being a Java process, requires a Java virtual machine (JVM) to run, and to support the Java applications running on it. As part of configuring an application server, you can fine-tune settings that enhance system use of the JVM. In addition to the following tuning parameters, see also “Java memory tuning tips” on page 163.

Use the following JVM parameters, including garbage collection options for IBM Developer Kit 1.4.2, to tune the Java virtual machine. For instructions on view and change the JVM configuration , go to “Using the JVM” on page 155. For information on specifying any of the following settings, go to “Java virtual machine settings” on page 155.

- **Specify any or all of the following generic JVM arguments.** These optional command line arguments are passed to the Java virtual machine code that starts the application server process.
  - Quickstart (-Xquickstart)
  - Avoiding class verification (-Xverify:none)
  - Class garbage collection (-Xnoclassgc)
  - Garbage collection threads (-Xgcthreads)
  - Garbage collection policy (-Xgcpolicy)
  - Sun JDK 1.4.2 Generational Garbage Collection (-XX)

    You can find more information about generational garbage collection at http://java.sun.com/docs/hotspot/gc/index.html.
  - Sun Java Development Kit 1.4.2 HotSpot JVM warm-up (-server)
  - Heap compaction (-Xnocompactgc)
  - Initial system heap size (-Xinitsh)
- **Set the initial heap size.**
- **Set the maximum heap size.**

Preparing to host applications

Rather than use the default application server provided with the product, you can configure a new server and set of resources.

The default application server and a set of default resources are available to help you begin quickly. You can choose instead to configure a new server and set of resources. Here is what you need to do in order to set up a run-time environment to support applications.

1. Create an application server.
2. Create a virtual host.
3. Configure a Web container.
4. Configure an EJB container.

5. Create resources for data access.
6. Create a JDBC provider and data source.
7. Create a URL and URL provider.
8. Create a JavaMail session.
9. Create resources for session support.
10. Configure a Session Manager.

Java memory tuning tips

Enterprise applications written in the Java language involve complex object relationships and utilize large numbers of objects. Although, the Java language automatically manages memory associated with object life cycles, understanding the application usage patterns for objects is important. In particular, verify the following:

- The application is not over-utilizing objects
- The application is not leaking objects
- The Java heap parameters are set properly to handle a given object usage pattern

Understanding the effect of garbage collection is necessary to apply these management techniques.

The garbage collection bottleneck

Examining Java garbage collection gives insight to how the application is utilizing memory. Garbage collection is a Java strength. By taking the burden of memory management away from the application writer, Java applications are more robust than applications written in languages that do not provide garbage collection. This robustness applies as long as the application is not abusing objects. Garbage collection normally consumes from 5% to 20% of total execution time of a properly functioning application. If not managed, garbage collection is one of the biggest bottlenecks for an application, especially when running on symmetric multiprocessing (SMP) server machines. The Java virtual machine (JVM) uses a parallel garbage collector to fully exploit an SMP during most garbage collection cycles where the Sun HotSpot 1.3.1 JVM has a single-threaded garbage collector. For more information about garbage collection in a Solaris operating environment see ″Performance: Resources for learning″ in the information center.

The garbage collection gauge

You can use garbage collection to evaluate application performance health. By monitoring garbage collection during the execution of a fixed workload, you gain insight as to whether the application is over-utilizing objects. Garbage collection can even detect the presence of memory leaks.

You can monitor garbage collection statistics using object statistics in the Tivoli Performance Viewer, or using the **verbose:gc** JVM configuration setting. The **verbose:gc** format is not standardized between different JVMs or release levels. For a description of the IBM verbose:gc output and more information about the IBM garbage collector, see ″Performance: Resources for learning″ in the information center.

For this type of investigation, set the minimum and maximum heap sizes to the same value. Choose a representative, repetitive workload that matches production usage as closely as possible, user errors included.

To ensure meaningful statistics, run the fixed workload until the application state is steady. It usually takes several minutes to reach a steady state.

Detecting over-utilization of objects

You can use the Tivoli Performance Viewer to check if the application is overusing objects, by observing the counters for the JVM runtime. You have to set the **-XrunpmiJvmpiProfiler** command line option, as well as the JVM module maximum level in order to enable the Java virtual machine profiler interface

(JVMPI) counters. The best result for the average time between garbage collections is at least 5-6 times the average duration of a single garbage collection. If you do not achieve this number, the application is spending more than 15% of its time in garbage collection.

If the information indicates a garbage collection bottleneck, there are two ways to clear the bottleneck. The most cost-effective way to optimize the application is to implement object caches and pools. Use a Java profiler to determine which objects to target. If you can not optimize the application, adding memory, processors and clones might help. Additional memory allows each clone to maintain a reasonable heap size. Additional processors allow the clones to run in parallel.

Detecting memory leaks

Memory leaks in the Java language are a dangerous contributor to garbage collection bottlenecks. Memory leaks are more damaging than memory overuse, because a memory leak ultimately leads to system instability. Over time, garbage collection occurs more frequently until the heap is exhausted and the Java code fails with a fatal `Out of Memory` exception. Memory leaks occur when an unused object has references that are never freed. Memory leaks most commonly occur in collection classes, such as Hashtable because the table always has a reference to the object, even after real references are deleted.

High workload often causes applications to crash immediately after deployment in the production environment. This is especially true for leaking applications where the high workload accelerates the magnification of the leakage and a memory allocation failure occurs.

The goal of memory leak testing is to magnify numbers. Memory leaks are measured in terms of the amount of bytes or kilobytes that cannot be garbage collected. The delicate task is to differentiate these amounts between expected sizes of useful and unusable memory. This task is achieved more easily if the numbers are magnified, resulting in larger gaps and easier identification of inconsistencies. The following list contains important conclusions about memory leaks:

- **Long-running test**

  Memory leak problems can manifest only after a period of time, therefore, memory leaks are found easily during long-running tests. Short running tests can lead to false alarms. It is sometimes difficult to know when a memory leak is occurring in the Java language, especially when memory usage has seemingly increased either abruptly or monotonically in a given period of time. The reason it is hard to detect a memory leak is that these kinds of increases can be valid or might be the intention of the developer. You can learn how to differentiate the delayed use of objects from completely unused objects by running applications for a longer period of time. Long-running application testing gives you higher confidence for whether the delayed use of objects is actually occurring.
- **Repetitive test**

  In many cases, memory leak problems occur by successive repetitions of the same test case. The goal of memory leak testing is to establish a big gap between unusable memory and used memory in terms of their relative sizes. By repeating the same scenario over and over again, the gap is multiplied in a very progressive way. This testing helps if the number of leaks caused by the execution of a test case is so minimal that it is hardly noticeable in one run.

  You can use repetitive tests at the system level or module level. The advantage with modular testing is better control. When a module is designed to keep the private module without creating external side effects such as memory usage, testing for memory leaks is easier. First, the memory usage before running the module is recorded. Then, a fixed set of test cases are run repeatedly. At the end of the test run, the current memory usage is recorded and checked for significant changes. Remember, garbage collection must be suggested when recording the actual memory usage by inserting System.gc() in the module where you want garbage collection to occur, or using a profiling tool, to force the event to occur.
- **Concurrency test**

  Some memory leak problems can occur only when there are several threads running in the application. Unfortunately, synchronization points are very susceptible to memory leaks because of the added complication in the program logic. Careless programming can lead to kept or unreleased references.

The incident of memory leaks is often facilitated or accelerated by increased concurrency in the system. The most common way to increase concurrency is to increase the number of clients in the test driver.

Consider the following points when choosing which test cases to use for memory leak testing:

- A good test case exercises areas of the application where objects are created. Most of the time, knowledge of the application is required. A description of the scenario can suggest creation of data spaces, such as adding a new record, creating an HTTP session, performing a transaction and searching a record.
- Look at areas where collections of objects are used. Typically, memory leaks are composed of objects within the same class. Also, collection classes such as Vector and Hashtable are common places where references to objects are implicitly stored by calling corresponding insertion methods. For example, the get method of a Hashtable object does not remove its reference to the retrieved object.

Tivoli Performance Viewer can help find memory leaks. For best results, repeat experiments with increasing duration, like 1000, 2000, and 4000-page requests. The Tivoli Performance Viewer graph of used memory should have a sawtooth shape. Each drop on the graph corresponds to a garbage collection. There is a memory leak if one of the following occurs:

- The amount of memory used immediately after each garbage collection increases significantly. The sawtooth pattern looks more like a staircase.
- The sawtooth pattern has an irregular shape.

Also, look at the difference between the number of objects allocated and the number of objects freed. If the gap between the two increases over time, there is a memory leak.

Heap consumption indicating a possible leak during a heavy workload (the application server is consistently near 100% CPU utilization), yet appearing to recover during a subsequent lighter or near-idle workload, is an indication of heap fragmentation. Heap fragmentation can occur when the JVM can free sufficient objects to satisfy memory allocation requests during garbage collection cycles, but the JVM does not have the time to compact small free memory areas in the heap to larger contiguous spaces.

Another form of heap fragmentation occurs when small objects (less than 512 bytes) are freed. The objects are freed, but the storage is not recovered, resulting in memory fragmentation until a heap compaction has been run.

To avoid heap fragmentation, turn on the -Xcompactgc flag in the JVM advanced settings command line arguments. The -Xcompactgc function verifies that each garbage collection cycle eliminates fragmentation. However, compaction is a relatively expensive operation. See the heap compaction command line argument (-Xnocompactgc) in “Java virtual machine settings” on page 155 for more information.

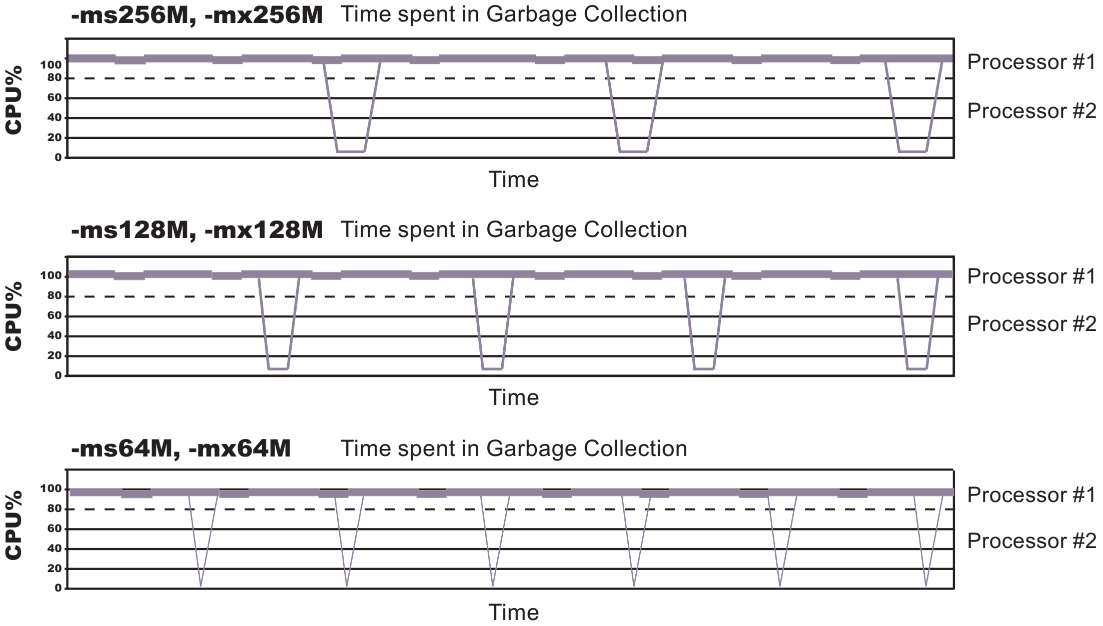
Java heap parameters

The Java heap parameters also influence the behavior of garbage collection. Increasing the heap size supports more object creation. Because a large heap takes longer to fill, the application runs longer before a garbage collection occurs. However, a larger heap also takes longer to compact and causes garbage collection to take longer. Refer to the sections on Java Heap sizes in “Java virtual machine settings” on page 155 for more information.

For performance analysis, the initial and maximum heap sizes should be equal.

When tuning a production system where the working set size of the Java application is not understood, a good starting value for the initial heap size is 25% of the maximum heap size. The JVM then tries to adapt the size of the heap to the working set size of the application.

Varying Java Heap Settings



The illustration represents three CPU profiles, each running a fixed workload with varying Java heap settings. In the middle profile, the initial and maximum heap sizes are set to 128MB. Four garbage collections occur. The total time in garbage collection is about 15% of the total run. When the heap parameters are doubled to 256MB, as in the top profile, the length of the work time increases between garbage collections. Only three garbage collections occur, but the length of each garbage collection is also increased. In the third profile, the heap size is reduced to 64MB and exhibits the opposite effect. With a smaller heap size, both the time between garbage collections and the time for each garbage collection are shorter. For all three configurations, the total time in garbage collection is approximately 15%. This example illustrates an important concept about the Java heap and its relationship to object utilization. There is always a cost for garbage collection in Java applications.

Run a series of test experiments that vary the Java heap settings. For example, run experiments with 128MB, 192MB, 256MB, and 320MB. During each experiment, monitor the total memory usage. If you expand the heap too aggressively, paging can occur. Use the **vmstat** command or the Windows NT or Windows 2000 Performance Monitor to check for paging. If paging occurs, reduce the size of the heap or add more memory to the system. When all the runs are finished, compare the following statistics:

- Number of garbage collection calls
- Average duration of a single garbage collection call
- Ratio between the length of a single garbage collection call and the average time between calls

If the application is not over-utilizing objects and has no memory leaks, the state of steady memory utilization is reached. Garbage collection also occurs less frequently and for short duration.

If the heap free space settles at 85% or more, consider decreasing the maximum heap size values because the application server and the application are under-utilizing the memory allocated for heap.

For more information about garbage collection see ″Performance: Resources for learning″ in the information center.

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.

Tuning application servers

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

This group of interrelated components is known as the “Queuing network” on page 27. The queuing network helps the system achieve maximum throughput while maintaining the overall stability of the system.

The follow steps describe various tuning tasks that may improve your application server performance. You can choose to implement any of these application server settings.

- **Tune the object request broker.** An Object Request Broker (ORB) manages the interaction between clients and servers, using the Internet InterORB Protocol (IIOP). It supports client requests and responses received from servers in a network-distributed environment. You can tune the ORB with the following parameters:
  - Set **Pass by reference (com.ibm.CORBA.iiop.noLocalCopies)** as described in “Object Request Broker service settings” on page 1816.
  - Set the **Connection cache minimum (com.ibm.CORBA.MaxOpenConnections)** as described in “Object Request Broker service settings” on page 1816.
  - Set **Maximum size** as described in “Thread pool settings” on page 120
  - Set **com.ibm.CORBA.ServerSocketQueueDepth** as described in “Object Request Broker custom properties” on page 1819.
  - Set the **com.ibm.CORBA.FragmentSize** as described in “Object Request Broker custom properties” on page 1819.

  The “Object Request Broker tuning guidelines” on page 1815 offer tips on using these parameters to tune the ORB.
- **Tune the XML parser definitions.**
  - **Description:** Facilitates server startup by adding XML parser definitions to the `jaxp.properties` and `xerxes.properties` files in the ${*install_root*}/jre/lib directory. The XMLParserConfiguration value might change as new versions of Xerces are provided.
  - **How to view or set:** Insert the following lines in both files:

    ```
    javax.xml.parsers.SAXParserFactory=org.apache.xerces.jaxp.SAXParserFactoryImpl
    javax.xml.parsers.DocumentBuildFactory=org.apache.xerces.jaxp.
              DocumentBuilderFactoryImpl
    org.apache.xerces.xni.parser.XMLParserConfiguration=org.apache.xerces.parsers.
             StandardParserConfiguration
    ```
  - **Default value:** None
  - **Recommended value:** None
- **Tune the dynamic cache service.** Using the dynamic cache service can improve performance. See “Task overview: Using the dynamic cache service to improve performance” on page 1901 for information about using the dynamic cache service and how it can affect your application server performance.
- **Tune the Web container.** The WebSphere Application Server Web container manages all HTTP requests to servlets, JSPs and Web services. Requests flow through a transport chain to the Web container. The transport chain defines the important tuning parameters for performance for the Web container. There is a transport chain for each TCP port that WebSphere Application Server is listening on for HTTP requests. For example, the default HTTP port 9080 is defined in Web container inbound channel chain. Use the following parameters to tune the Web container:
  - HTTP requests are processed by a pool of server threads. The minimum and maximum thread pool size for the Web container can be configured for optimal performance. Generally, 5 to 10 threads per server CPU will provide the best throughput. The number of threads configured does not represent the number of requests WebSphere can process concurrently. Requests are queued in the transport chain when all threads are busy. To specify the thread pool settings:

    1. Click **Servers > Application Servers >***server_name* **Web Container Settings> Web Container > Web container transport chains**.
    2. Select the normal inbound chain for serving requests. This will usually be named WCInboundDefault, on port 9080.
    3. Click **TCP Inbound Channel (TCP_2)**.
    4. Set **Thread Pools** under Related Items.
    5. Select **WebContainer**.
    6. Enter values for **Minimum Size** and **Maximum Size**.
  - The HTTP 1.1 protocol provides a ″keep-alive″ feature to enable the TCP connection between HTTP clients and the server to remain open between requests. By default WebSphere Application Server will close a given client connection after a number of requests or a timeout period. After a connection is closed, it will be recreated if the client issues another request. Early closure of connections can reduce performance. Enter a value for the maximum number of persistent requests to (keep-alive) to specify the number of requests that are allowed on a single HTTP connection. Enter a value for persistent timeouts to specify the amount of time, in seconds, that the HTTP transport channel allows a socket to remain idle between requests. To specify values for Maximum persistent requests and Persistent timeout:
    1. Click **Servers > Application Servers >***server_name* **Web Container Settings> Web Container > Web container transport chains**.
    2. Select the normal inbound chain for serving requests. This will usually be named WCInboundDefault, on port 9080.
    3. Click **HTTP Inbound Channel (HTTP_2)**.
    4. Enter values for **Maximum persistent requests** and **Persistent timeout**.

- **Tune the EJB container.** An EJB container is automatically created when you create an application server. After the EJB container is deployed, you can use the following parameters to make adjustments that improve performance.
  - Set the **Cleanup interval** and the **Cache size** as described in “EJB cache settings” on page 834.
  - **Break CMP enterprise beans into several enterprise bean modules** while assembling EJB modules. See ″Assembling EJB modules″ in the information center for more information.

  See also “EJB method Invocation Queuing” on page 820.

- **Tune the session management.** The installed default settings for session management are optimal for performance. See “Tuning session management” on page 797 and “Tuning parameter settings” on page 798 for more information about tuning session management.

- **Tune the data sources.** A data source is used to access data from the database. The following parameters reveal how the number of physical connections within a connection pool can change performance.
  - Review information on “Connection pooling” on page 1137.
  - Set the **Maximum connection pool** and **Minimum connection pool** as described in “Connection pool settings” on page 1199.
  - Set the **Statement cache size** as described in “Data source settings” on page 1173.

Web services client to Web container optimized 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 direct communication eliminates the need for clients and web containers that are in the same process to communicate over the network. For example, a Web services client might be running in an application server. Instead of accessing the network to communicate with the Web container, the Web services client can communicate with the Web container using the optimized local path. This optimized local path improves the performance of the application server by allowing Web services clients and Web containers to communicate without using network transports.

In a clustered environment, there is typically an HTTP server (such as IBM HTTP server) that handles incoming client requests, distributing them to the correct application server in the cluster. The HTTP server uses information about the requested application and the defined virtual hosts to determine which application server receives the request. The Web services client also uses the defined virtual host information to determine whether the request can be served by the local Web container. You must define unique values for the host and port on each application server. You cannot define the values of host and port as wild cards denoted by the asterisk symbol (*) when you enable the optimized communication between the Web services application and the Web container. Using wild cards indicate that the local Web container can handle Web services requests for all destinations.

The optimized local communication path is disabled by default. You can enable the local communication path with the `enableInProcessConnections` custom property. Before configuring this custom property, make sure that you are not using wild cards for host names in your Web container end points. Set this property to **true** in the Web container to enabled the optimized local communication path. When disabled, the Web services client and the Web container communicate using network transports.

For information about how to configure the `enableInProcessConnections` custom property, see the *Developing and deploying applications* PDF.

When the optimized local communication path is enabled, logging of requests through the local path uses the same log attributes as the network channel chain for the Web container. To use a different log file for in process requests than the log file for network requests, use a custom property on the HTTP Inbound Channel in the transport chain. Use the `inProcessLogFilenamePrefix` custom property to specify a string that is added to the beginning of the network log file name to create a file name that is unique. Requests through the local process path are logged to this specified file. For example, if the log filename is `../httpaccess.log` for a network chain, and the `inProcesslLogFilenamePrefix` custom property is set to “local” on the HTTP channel in that transport chain, the local log file name for requests to the host associated with that chain is `/localhttpaccess.log`.

Application servers: Resources for learning

Use the following links to find relevant supplemental information about configuring application servers. 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 instructions and examples
- Programming specifications
- Administration

Programming instructions and examples

- WebSphere Application Server education at http://www.ibm.com/software/webservers/learn/.

Programming specifications

- The Java^{TM} Virtual Machine Specification, Second Edition at http://java.sun.com/docs/books/vmspec/.
- Sun’s technology forum for the Java^{TM} Virtual Machine Specification at http://forum.java.sun.com/forum.jsp?forum=37

Administration

- Listing of all IBM WebSphere Application Server Redbooks at http://publib-b.boulder.ibm.com/Redbooks.snf/Portals/WebSphere.

Chapter 4. Using the administrative clients

The product provides a variety of administrative clients for deploying and administering your applications and application serving environment, including configurations and logical administrative domains.

- "Using the administrative console"

  The administrative console is a graphical, browser-based tool.
- "Getting started with scripting" on page 186

  Scripting is a non-graphical alternative that you can use to configure and administer your applications and application serving environment. The WebSphere Application Server **wsadmin** tool provides the ability to run scripts. The wsadmin tool supports a full range of product administrative activities.
- "Using Ant to automate tasks" on page 626

  To support using Apache Ant with Java 2 Platform, Enterprise Edition (J2EE) applications running on IBM WebSphere Application Server, the product provides a copy of the Ant tool and a set of Ant tasks that extend the capabilities of Ant to include product-specific functions.
- "Using administrative programs (JMX)" on page 627

  The product supports access to the administrative functions through a set of Java classes and methods, under the Java Management Extensions (JMX) specification. You can write a Java program that performs any of the administrative features of the other administrative clients. You also can extend the basic product administrative system to include your own managed resources.
- "Using command line tools" on page 668

  Several command-line tools are available that you can use to start, stop, and monitor WebSphere server processes and nodes. These tools work on local servers and nodes only. They cannot operate on a remote server or node.

Using the administrative console

The administrative console is a Web-based tool that you use to manage the IBM WebSphere Application Server product as well as the Network Deployment product. The administrative console supports a full range of product administrative activities.

1. Distributed platforms: Start the server for the administrative console.
2. Access the administrative console.
3. Change the session timeout for the administrative console. (Optional)
4. Browse the administrative console.
5. Specify console preferences.
6. Access help.

Starting and stopping the administrative console

This topic describes how to set up the administrative console environment, to access the administrative console, and to log out of the administrative console.

To access the administrative console, you must start it and then log in. After you finish working in the console, save your work and log out.

1. Start the administrative console.
   a. Distributed platforms: Verify that the administrative console runs on the `server1` application server for the WebSphere base product.
   b. Enable cookies in the Web browser that you use to access the administrative console for the administrative console to work correctly.
   c. Distributed platforms: In the same Web browser, type `http://`*`your_fully_qualified_server_name`*`:9060/ibm/console`, where

your_fully_qualified_server_name is the fully qualified host name for the machine that contains the administrative server. When the administrative console is on the local machine, *your_fully_qualified_server_name* can be `localhost` unless security is enabled. On Windows platforms, use the actual host name if `localhost` is not recognized. If security is enabled, your request is redirected to `https://`*`your_fully_qualified_server_name`*`:9043/ibm/console`, where *your_fully_qualified_server_name* is the fully qualified host name for the machine that contains the administrative server.

For a listing of supported Web browsers, see WebSphere Application Server system requirements at

```
http://www.ibm.com/software/webservers/
appserv/doc/latest/prereq.html
```

The Web address appears on two lines for printing purposes. Enter the Web address on one line in your browser.

d. Wait for the console to load into the browser. A Login page is displayed after the console starts.

If you cannot start the administrative console because the console port conflicts with an application that is already running on the machine, change the port number in the *`install_root`*`/profiles/`*`profile name`*`/config/cells/`*`cell_name`*`/ nodes/`*`node_name`*`/servers/`*`server_name`*`/server.xml` file and the *`install_root`*`/profiles/`*`profile name`*`/config/cells/`*`cell_name`*`/virtualhosts.xml` files. Change all the occurrences of port 9060 (or the port that is selected during profile creation for WebSphere Application Server) to the port for the console. Alternatively, shut down the other application that uses the conflicting port before starting the WebSphere Application Server product.

2. Log into the console.

   a. Enter your user name or user ID.

      The user ID lasts only for the duration of the session for which it was used to log in.

      Changes made to server configurations are saved to the user ID. Server configurations also are saved to the user ID if a session timeout occurs.

      If you enter an ID that is already in use (and in session), you are prompted to do one of the following actions:

      - Force the existing user ID out of session. The configuration file that is used by the existing user ID is saved in the temporary area.
      - Wait for the existing user ID to log out or time out of the session.
      - Specify a different user ID.

   b. If the console is secure, you must also enter a password for the user name. The console is secure if someone has taken the following actions for the console:

      - Specified security user IDs and passwords
      - Enabled global security

   c. Click **OK**.

3. Stop the administrative console. Click **System administration > Save changes to Master Repository > Save** to save work. Then click **Logout** to exit the console.

   If you close the browser before saving your work, when you next log in under the same user ID, you can recover any unsaved changes.

Login settings

Use this page to specify the user for the WebSphere Application Server administrative console. If you are using global security, then you must also specify a password.

When you specify a user, you can resume work done previously with the product. After you type in a user ID, and password if you are using global security, click **OK** to proceed to the next page and access the administrative console.

To view this page, start the administrative console.

Logging into the administrative console: When you log into the administrative console, you can optionally specify a user ID if the console is not secure. If the administrative console is secure, you must specify a user ID and password.

User ID

Specifies a string that identifies the user. The user ID must be unique to the administrative server. Concurrent administrative console sessions must use unique user IDs.

Work that you do with the product and then save before exiting the product is saved to a configuration that is identified by the user ID that you enter. To later access work done under that user ID, specify the same user ID in the Login page.

Data type	String

Password

If you use global security, specify a password.

Resolving conflicts during login: Conflicts can result if you log into the administrative console with a user ID that is already in use.

Another user is currently logged in with the same user name

Specifies whether to log out the user and to continue work with the user ID that is specified, or to return to the Login page and specify a different user ID, or wait for the user to log out.

This field is displayed if:

- The user closed a Web browser while browsing the administrative console and did not first log out, then opened a new browser and tried to access the administrative console with the same user ID.
- The user opened a Web browser to access the administrative console while accessing the administrative console in another open Web browser with the same user ID.
- The user opens a Web browser and attempts to log into the console with the same user ID that is already in use by another user who logged into the console from another Web browser on another computer.

Recovering prior changes: You can either recover changes that you made to the configuration from a prior session or use the master configuration. The default is to recover changes from a prior session.

Recover changes made in a prior session

When enabled, this setting specifies that you want to use the same administrative configuration used for the last user's session. This option recovers changes made by the user since the last saving of the administrative configuration for the user's session.

This field is displayed only if the user changed the administrative configuration and then logged out without saving the changes.

Work with the master configuration

When enabled, this setting specifies to use the default administrative configuration instead of the configuration that was last used for the user's session. Changes that are made to the user's session since the last saving of the administrative configuration are lost.

This field is displayed only if the user changed the administrative configuration and then logged out without saving the changes.

Resolving login failures: When the administrative console is enabled with global security, you must type in a valid user ID and password. If the user ID, password, or both are not valid, you receive the following message:

```
Unable to process login. Please check User ID and password and try again.
```

Resolve the problem by entering a valid user ID and password as defined in the WebSphere Application Server security documentation.

Save changes to the master configuration

Use this page to update the master repository with your administrative console changes, to discard your administrative console changes and continue working with the master repository, or to continue working with your administrative console changes that are not saved to the master repository.

Until you save changes to the master repository, the administrative console uses a local workspace to track your changes.

Total changed documents: Specifies the total number of documents that you changed for your session, but that are not saved to the master repository. By clicking the +/- toggle key, you can see additional information about the changed documents:

- **Changed items**

  When you change your local configuration, each path and configuration file that you can apply the update to in the master repository is displayed in the list.
- **Status**

  Can contain the following options:
  - **Added:** If you save your changes to the master repository, a new configuration file is created on the indicated path.
  - **Updated:** If you save your changes to the master repository, an existing configuration file is updated on the indicated path.
  - **Deleted:** If you save your changes to the master repository, an existing configuration file is deleted on the indicated path.

Synchronize changes with nodes: Specifies whether you want to force node synchronization at the time that you save your changes to the master repository rather than when node synchronization normally occurs.

Setting the session timeout for the administrative console

This topic describes how to change the session timeout from the default value for the administrative console.

Ensure that you have the proper permissions to change the `${WAS_HOME}/systemApps/adminconsole.ear/deployment.xml` file.

Determine whether the default session timeout value of 30 minutes is acceptable. Some reasons that you might change the default value are:

- Users in secure environments might need shorter session timeout periods to ensure security, encase they leave their machine and forget to log off the console.
- Users might need longer session timeout periods if they respond slower than typical users for accessibility reasons.
- Users in secure environments might not want the administrative console timeout value to conflict with Lightweight Third-Party Authentication (LTPA) cookie timeouts

Do the following actions to change the timeout value:

1. Edit the `${WAS_HOME}/systemApps/adminconsole.ear/deployment.xml` file in a text editor.
2. Locate the xml statement `<tuningParams xmi:id="TuningParams_1088453565469" maxInMemorySessionCount="1000" allowOverflow="true" writeFrequency="TIME_BASED_WRITE" writeInterval="10" writeContents="ONLY_UPDATED_ATTRIBUTES" invalidationTimeout="30">`
3. Change the invalidationTimeout value to the desired session timeout. The default is 30.
4. Save the `${WAS_HOME}/systemApps/adminconsole.ear/deployment.xml` file.
5. Restart the console.

Once you restart the console, the change takes effect.

Manage WebSphere Application Server through the administrative console.

Administrative console areas

Use the administrative console to create and manage objects in the WebSphere Application Server configuration such as resources, applications, and servers. Additionally, use the administrative console to view product messages. This topic describes the main areas that display on the administrative console.

To view the administrative console, ensure that the application server for the administrative console is running. Point a Web browser at the Web address for the administrative console, enter your user ID and, if needed, a password on the Login page.

You can resize the width of the navigation tree and workspace simultaneously by dragging the border between them to the left or the right. The change in width does not persist between administrative console user sessions.

The console has the following main areas.

Taskbar

The taskbar offers options for logging out of the console, accessing product information, and accessing support.

Navigation tree

The navigation tree on the left side of the console offers links to console pages that you use to create and manage components in a WebSphere Application Server administrative cell.

Click a plus sign (**+**) beside a tree folder or item to expand the tree for the folder or item. Click a minus sign (**-**) to collapse the tree for the folder or item. Click an item in the tree view to toggle its state between expanded and collapsed.

Workspace

The workspace on the right side of the console contains pages that you use to create and manage configuration objects such as servers and resources. Click links in the navigation tree to view the different types of configured objects. Within the workspace, click configured objects to view their configurations, run-time status, and options. Click **Welcome** in the navigation tree to display the workspace Home page, which contains links to information on using the WebSphere Application Server product.

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.

- **Check all.** Selects each resource that is listed on the administrative console panel, in preparation for performing an action against the selected resources.
- **Uncheck all.** Removes all the listed resources from each selection so that no action is performed against any of the resources.

- **Filter the view.** Produces a dialog box for specifying the resources to view in the table on this administrative console page.

  Hide the filter view. Hides the dialog box for specifying the resources to view in the table on this administrative console page.

  When you produce the dialog box, select the column to filter and enter the filter criteria.

  Column to filter

  Select the column to filter from the drop-down list. When you apply the filter, only those items in the selected column that meet the filter criteria are displayed.

  For example, select **Names** to enter criteria by which to filter application server names.

  Filter criteria

  Enter a string that must be found in the name of a collection entry to qualify the entry to display in the collection table. The string can contain percent sign (%), asterisk (*), or question mark (?) symbols as wildcard characters. For example, enter `*App*` to find any application server whose name contains the string `App`.

  Prefix each of the following characters `( ) ^ * % { } \ + $` with a backslash (\) so that the regular expression engine performing the search correctly matches the search criteria. For example, to search for all Java DataBase Connectivity (JDBC) providers containing `(XA)` in the provider name, specify the following string:

  ```
  *\(XA\)
  ```
- **Clear filter criteria.** Clears your filter changes and restores the most recently saved values.
- **Abort.** Stops a transaction that is not yet in the prepared state. All operations that the transaction completed are undone.
- **Activate.** Activates a group member.
- **Add.** Adds the selected or typed item to a list, or produces a dialog for adding an item to a list.
- **Apply.** Saves your changes to a page without exiting the page.
- **Back.** Displays the previous page or item in a sequence. The administrative console does not support using the Back and Forward options of a browser, which can cause intermittent problems. Use Back or Cancel on the administrative console panels instead.
- **Balance.** Balances active members in high availability groups across servers that host the high availability groups. The administrator must first determine which groups have active members and select those groups before selecting Balance.
- **Browse.** Opens a dialog that enables you to look for a file on your system.
- **Calculate groups.** Calculates the number of high availability groups that are returned based on the match set.
- **Cancel.** Exits the current page or dialog, discarding unsaved changes. The administrative console does not support using the Back and Forward options of a browser, which can cause intermittent problems. Use Cancel on the administrative console panels instead.
- **Change.** In the context of security, you can search the user registry for a user ID for an application to run under. In the context of container properties, you can change the data source that the container is using.
- **Clear.** Clears your changes and restores the most recently saved values.
- **Clear selections.** Clears any selected cells in the tables on this tabbed page.
- **Close.** Exits the dialog.
- **Commit.** Releases all locks that are held by a prepared transaction and forces the transaction to commit.
- **Copy.** Creates copies of the selected application servers.
- **Create.** Saves your changes to all the tabbed pages in a dialog and exits the dialog.
- **Create tables.** Develops scheduler database tables.
- **Deactivate.** Deactivates a group member. The group member must be in the active state to be deactivated. The deactivate option causes the group member to move to the idle state. The group policy overrides which members are activated and deactivated for a group. The policy is enforced for every member state change. If the deactivate option conflicts with the group policy, the policy resets who is the active member of the group.
- **Delete.** Removes the selected instance.

- **Details.** Shows the details about a transaction.
- **Disable.** Disables a group or group member. When you disable a group or group member, the active group or group member is first deactivated. If the deactivate option is successful, the group or group member moves to the disable state. A disabled group or group member cannot be activated.
- **Done.** Saves your changes to all the tabbed pages in a dialog and exits the dialog.
- **Down.** Moves through a list.
- **Drop tables.** Removes scheduler database tables.
- **Dump.** Activates a dump of a traced application server.
- **Edit.** Lets you edit the selected item in a list, or produces a dialog box for editing the item.
- **Enable.** Enables a group or a group member.
- **Export.** Accesses a page for exporting enterprise archive (EAR) files for an enterprise application.
- **Export DDL.** Accesses a page for exporting data definition language (DDL) files for an enterprise application.
- **Export Keys.** Exports Lightweight Third-Party Authentication (LTPA) keys to other domains.
- **Export route table.** Exports the route table information for a selected cluster to a binary file in the configuration.
- **Filter.** Produces a dialog box for specifying the resources to view in the tables on this tabbed page.
- **Finish.** Forces a transaction to finish, regardless of whether its outcome has been reported to all participating applications.
- **First.** Displays the first record in a series of records.
- **Generate keys.** Generates new LTPA keys. When security is turned on for the first time with LTPA as the authentication mechanism, LTPA keys are automatically generated with the password entered in the panel. To generated new keys, use this option after the server is up with security turned on. Clicking this option generates the keys and propagates them to all active servers (cell, node, and application servers). The new keys can be used to encrypt and decrypt the LTPA tokens. Click **Save** on the console taskbar to save the new keys and the password in the repository.
- **Immediate stop.** Stops the server, but bypasses the normal server quiesce process that supports in-flight requests to complete before shutting down the entire server process. This shutdown mode is faster than the normal server stop processing, but some application clients can receive exceptions.
- **Import keys.** Imports new LTPA keys from other domains. To support single signon (SSO) in WebSphere Application Server across multiple WebSphere domains (cells), share LTPA keys and a password among the domains. After exporting the keys from one of the cells into a file, click this option to import the keys into all the active servers (cell, node, and application servers). The new keys can be used to encrypt and decrypt the LTPA token. Click **Save** on the console taskbar to save the new keys and the password in the repository.
- **Install.** Displays the Preparing for application installation page, which you use to deploy an application, an enterprise bean, or a Web component onto an application server.
- **Install RAR.** Opens a dialog that is used to install a Java 2 Platform, Enterprise Edition Connector Architecture (JCA) connector and to create a resource adapter.
- **Manage transactions.** Displays a list of active transactions running on a server. You can forcibly finish any transaction that has stopped processing because a transactional resource is not available.
- **Modify.** Opens a dialog that is used to change a specification.
- **Move.** Moves the selected application servers to a different location in the administrative cell. When prompted, specify the target location.
- **Move down.** Moves downward through a list.
- **Move up.** Moves upward through a list.
- **New.** Displays a page that you use to define a new instance. For example, clicking **New** on the Application Servers page displays a page on which you can configure a new application server.
- **Next.** Displays the next page, frame, or item in a sequence.
- **OK.** Saves your changes and exits the page.
- **Ping.** Attempts to contact selected application servers.
- **Previous.** Displays the previous page, frame, or item in a sequence.
- **Quit.** Exits a dialog box and discards any unsaved changes.
- **Refresh.** Refreshes the view of data for instances that are currently listed on this tabbed page.
- **Remove.** Deletes the selected item.
- **Remove file.** Removes the specified file from the selected application or module.

- **Reset.** Clears your changes on the tab or page and restores the most recently saved values.
- **Restart.** Stops the selected objects and starts them again.
- **Retrieve new.** Retrieves a new record.
- **Rollout update.** Sequentially updates an application that is installed on multiple cluster members across a cluster. After you update application files or a configuration, click **Rollout update** to install the configuration or the updated files for an application on all the cluster members of a cluster on which the application is installed. The Rollout update option applies the following steps to each cluster member in sequence:
  1. Saves an updated configuration.
  2. Stops the cluster member.
  3. Updates the application on the node by synchronizing the configuration.
  4. Restarts the cluster member.

  This action enables you to update an application on multiple cluster members while providing continuous availability of the application.
- **Save.** Saves the changes in your local configuration to the master configuration.
- **Select.** For resource analysis, lets you select a scope in which to monitor resources.
- **Set.** Saves your changes to settings in a dialog.
- **Settings.** Displays a dialog for editing servlet-related resource settings.
- **Settings in use.** Displays a dialog showing the settings in use.
- **Show groups.** Displays a collection of high availability groups, based on the match set.
- **Show servers.** Displays a collection of servers that are contained in the high availability groups that match the match set.
- **Start.** In the context of application servers, starts selected application servers. In the context of data collection, starts collecting data for the tables on this tabbed page.
- **Stop.** In the context of server components such as application servers, stops the selected server components. In the context of a data collection, stops collecting data for the tables on a tabbed page.
- **Terminate.** Deletes the Application Server process or another process that cannot be stopped by the **Stop** or **Immediate Stop** commands. Some application clients can receive exceptions. Always attempt an immediate stop before using this option.
- **Uninstall.** Deletes a deployed application from the WebSphere Application Server configuration repository. Also deletes application binary files from the file system.
- **Update.** Replaces an application that is deployed on a server with an updated application. As part of the updating, you might need to complete steps on the Preparing for application installation and Update application pages.
- **Update resource list.** Updates the data on a table. Discovers and adds new instances to the table.
- **Verify tables.** Validates the mapping between the table names, scheduler resource, and data sources.
- **View.** Opens a dialog on a file.

Administrative console page features

This topic provides information about the basic elements of an administrative console page, such as the various tabs.

Administrative console pages are arranged in a few basic patterns. Understanding their layout and behavior will help you use them more easily.

Collection pages

Use collection pages to manage a collection of existing administrative objects. A collection page typically contains one or more of the following elements:

Scope and Preferences

These are described in “Administrative console scope settings” on page 182 and “Administrative console preference settings” on page 182.

Table of existing objects

The table displays existing administrative objects of the type specified by the collection page. The table columns summarize the values of the key settings for these objects. If no objects exist yet, an empty table is displayed. Use the available buttons to create a new object.

Buttons for performing actions
The available buttons are described on the Administrative console buttons help panel. In most cases, you need to select one or more of the objects in the table, then click a button. The action will be applied to the selected objects.

Sort toggle buttons
Following column headings in the table are icons for sort ascending (^) and sort descending (v). By default, items such as names are sorted in descending order (alphabetically). To enable another sorting order, click on the icons for the column whose items you want sorted.

Detail pages

Use detail pages to configure specific administrative objects, such as an application server. A detail page typically contains one or more of the following elements:

Configuration tabbed page
This tabbed page is for modifying the configuration of an administrative object. Each configuration page has a set of general properties specific to the administrative object. Other sets of properties display on the page, but vary depending on the administrative object.

Runtime tabbed page
This tabbed page displays the configuration that is currently in use for the administrative object. It is read-only in most cases. Some detail pages do not have runtime tabs.

Local Topology tabbed page
This tabbed page displays the topology that is currently in use for the administrative object. View the topology by expanding and collapsing the different levels of the topology. Some detail pages do not have local topology tabs.

Buttons for performing actions
Buttons to perform specific actions display on the configuration tabbed page and the runtime tabbed page. The displayed buttons vary based on the administrative object. The available buttons are described on the Administrative console buttons help panel.

Wizard pages

Use wizard pages to complete a configuration process comprised of several steps. Be aware that wizards show or hide certain steps depending on the characteristics of the specific object you are configuring.

Administrative console navigation tree actions

Use the navigation tree of the administrative console to access pages for creating and managing servers, applications, resources, and other components.

To view the navigation tree, go to the WebSphere Application Server administrative console and look at the tree on the left side of the console. The tree provides navigation to configuration tasks and run-time information. The main topics available on the navigation tree are detailed in the following section. To use the tree, expand a main topic and select an item from the expanded list to display a page on which you can perform the administrative task.

Servers:

Configure application servers, clusters, generic servers, Web servers, and core groups.

Applications:

Install applications onto servers and manage the installed applications.

Resources:

Configure resources and to view information on resources that exist in the administrative cell.

Security:

Access the Security Center, which you use to secure applications and servers.

Environment:

Configure hosts, WebSphere Application Server variables, and other components.

System Administration:

Configure console settings, and manage components and users of a Network Deployment product.

Troubleshooting:

Check for configuration errors and problems, view log files, and enable and disable tracing on a distributed platform.

Monitoring and Tuning:

Monitor and tune your Application Server performance and analyze performance data.

Service Integration:

Iimplement message-oriented and service-oriented applications.

UDDI:

Publish and discover information about Web services.

Administrative console taskbar actions

Use the taskbar of the administrative console to log out of the administrative console and to access the console help.

To view the taskbar, go to the WebSphere Application Server administrative console and look at the horizontal bar near the top of the console. The taskbar provides the following actions.

Logout: Logs you out of the administrative console session and displays the Login page. If you made changes to the administrative configuration since last saving the configuration to the master repository, the Save page is displayed before returning to the Login page.

- Click **Save** to save the changes to the master repository.
- Click **Discard** to exit the session without saving changes.
- Click **Logout** to exit the session without saving changes but with the opportunity to recover your changes when you return to the console.

Help: Opens a new Web browser to online help for the WebSphere Application Server product.

Support: Displays support links that vary based on the products that extend the WebSphere Application Server. Use the support page to access product information such as Frequently Asked Questions (FAQs), technical notes (Technotes), hints and tips, and news. You can additionally install the Support Advisor Search application so that when you click on the support link, a new Web browser that contains the Support Advisor Search application opens. The Support Advisor Search application displays the support links on the support page, but additionally provides federated search capabilities into IBM knowledge databases.

Specifying console preferences

Use this topic to customize how much data displays on an administrative console panel.

Throughout the administrative console are pages that have Preferences fields, Scope fields, and Filter radio buttons. By selecting these fields and radio buttons you can customize how much data is shown.

For example, examine the Preferences field for the Enterprise Applications page:

1. Go to the navigation tree of the administrative console and click **Applications > Enterprise Applications**.
2. Expand **Preferences**.
3. For the **Maximum rows** field, specify the maximum number of rows to display when the collection is large. The default is 20. Rows that exceed the maximum number display on subsequent pages.
4. Select **Retain filter criteria** if you want to retain the last filter criteria that is entered in the filter function. When you return to the Applications page, the page initially uses the retained filter criteria to display the collection of applications in the table following the preferences. Otherwise, clear **Retain filter criteria** and the last filter criteria is not retained.
5. Click **Apply** to apply your selections or click **Reset** to return to the default values. The default is not to enable (not have a check mark beside) **Retain filter criteria**.

Other pages have similar fields and radio buttons that you can use to specify console preferences. While Preferences fields, Scope fields, and Filter buttons control how much data is shown in the console, the **Preferences** option controls general behavior of the console. Click **System administration > Console settings > Preferences** to view the Preferences page.

Preferences settings

Use the Preferences page to specify whether you want the administrative console workspace to refresh automatically after changes, the default scope to be the administrative console node, confirmation dialogs to display, and the workspace banner and descriptions to display.

To view this administrative console page, click **System administration > Console settings > Preferences**.

Turn on workSpace auto-refresh:

Specifies whether you want the administrative console workspace to redraw automatically after the administrative configuration changes.

The default is for the workspace to redraw automatically. If you direct the console to create a new instance of, for example, an application server, the Application Servers page refreshes automatically and shows the new server name in the collection of servers.

Specifying that the workspace not redraw automatically means that you must access a page again by clicking the console navigation tree or links on collection pages to see the changes that are made to the administrative configuration.

Default	true (selected)

No confirmation on workspace discard:

Specifies whether the confirmation dialog is displayed after a request is receive to discard the workspace. The default is to display confirmation dialogs.

Default	false (cleared)

Use default scope (administrative console node):

Specifies whether the default scope is the administrative console node. The default scope not is not the console node.

Default	false (cleared)

Show banner:

Specifies whether the WebSphere Application Server banner along the top of the administrative console is displayed. The default is for the banner to display.

Default	true (selected)

Show Descriptions:

Specifies whether information on the right of the console is shown. The default is to show the information.

Data type	Boolean
Default	true

Administrative console preference settings

Use the preference settings to specify how you want information displayed on an administrative console page.

Maximum rows: Indicates the maximum number of rows to display per page when the collection is large.

Filter history: Indicates whether to use the same filter criteria to display this page the next time you visit it.

Select the **Retain filter criteria** check box to retain the last filter criteria entered. When you return to the page, retained filter criteria control the application collection that is displayed n the table.

Show confirmation for stop command: Select the check box if you want a confirmation that the **stop** command is successful.

Show confirmation for immediate stop command: Select the check box if you want a confirmation that the **immediate stop** command is successful.

Show confirmation for terminate command: Select the check box if you want a confirmation that the **terminate** command is successful.

Administrative console scope settings

Use this page to specify the level at which a resource is visible on the administrative console panel. A resource can be visible in the administrative console collection table at the cell, node, cluster, or server scope. By changing the value for Scope you can see other variables that apply to a resource and might change the contents of the collection table.

Click **Browse** next to a field to see choices for limiting the scope of the field. If a field is read-only, you cannot change the scope. For example, if only one server exists, you cannot switch the scope to a different server.

You always create resources at the current scope that is selected in the administrative console panel, even though the resources might be visible at more than one scope.

Resources such as JDBC providers, namespace bindings, or shared libraries can be defined at multiple scopes. Resources that are defined at more specific scopes override duplicate resources that are defined at more general scopes.

- The application scope has precedence over all the scopes.
- The server scope has precedence over the node, cell, and cluster scopes.
- The cluster scope has precedence over the node and cell scopes.
- The node scope has precedence over the cell scope.

Despite the scope of a defined resource, the resource properties only apply at an individual server level. For example, if you define the scope of a data source at the cell level, all the 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 the maximum connections as 10, then each server in that cell can have 10 connections.

The cell scope is the most general scope and does not override any other scope. The recommendation is that you generally specify a more specific scope than the cell scope. When you define a resource at a more specific scope, you provide greater isolation for the resource. When you define a resource at a more general scope, you provide less isolation. Greater exposure to cross-application conflicts occur for a resource that you define at a more general scope.

Cell Limits the visibility to all servers on the named cell. The resource factories within the cell scope are:

- Defined for all servers within this cell
- Overridden by any resource factories that are defined within application, server, cluster and node scopes that are in this cell and have the same Java Naming and Directory Interface (JNDI) name

The resource providers that are required by the resource factories must be installed on every node within the cell before applications can bind or use them.

Cluster

Limits the visibility to all the servers on the named cluster. All cluster members must at least be at Version 6 to use cluster scope for the cluster. The resource factories that are defined within the cluster scope:

- Are 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

The resource factories that are defined within the cell scope are available for this cluster to use, in addition to the resource factories, that are defined within this cluster scope.

Node Limits the visibility to all the servers on the named node. The node scope is the default scope for most resource types. The resource factories that are defined within the node scope:

- Are available for servers on this node to use
- Override any resource factories that have the same JNDI name defined within the cell scope

The resource factories that are defined within the cell scope are available for servers on this node to use, in addition to the resource factories that are defined within this node scope.

Server

Limits the visibility to the named server. The server scope is the most specific scope for defining resources. The resource factories that are defined within the server scope:

- Are available for applications that are deployed on this server
- Override any resource factories that have the same JNDI name defined within the node and cell scopes

The resource factories that are defined within the node and cell scopes are available for this server to use, in addition to the resource factories that are defined within this server scope.

Application

Limits the visibility to the named application. Application scope resources cannot be configured from the console. Use the WebSphere Application Server Toolkit (AST) or the wsadmin tool to view or modify the application scope resource configuration. The resource factories that are defined within the application scope are available for this application to use only. The application scope overrides all other scopes.

You can configure resources and WebSphere Application Server variables under all five scopes. You can configure namespace bindings and shared libraries only under cell, node, and server scopes.

Accessing help and product information from the administrative console

This topic describes how to use administrative console help and how to link to product documentation from the administrative console.

You must have a connection to the Internet to access information about WebSphere Application Server from the Welcome page of the administrative console.

All of the helps panels that you can access from the administrative console, you can access from the WebSphere Application Server Information Center. This article describes how to access the help panels, the information center, and other product documentation from the administrative console.

- Click **Welcome** on the administrative console navigation tree. In the workspace to the right of the navigation tree, select the appropriate links to access the WebSphere Application Server Information Center, the WebSphere Application Server product information, and the WebSphere Application Server technical information on developerWorks.
- Access help in the following ways:
  - Click **Help** on the administrative console task bar to open a new Web browser for online help.
    - Click on the **Help index** tab and select from the list of help panels to view administrative console help information.
    - Click on the **Search** tab, provide search terms, and then click **Search**. Under Results, select a help panel that contains the search information.
  - Click the **?** icon on the task bar for the particular administrative console panel to open a new Web browser and view the help panel for the corresponding administrative console panel. The help panel is displayed in the Help index for the administrative console.
  - In the help portal that is on the right side of the administrative console panel, do one or all of the following tasks:
    - Click a field label or a list marker in the administrative console panel for the help to display under Field help. Alternatively, place the cursor over the field label or the list marker for the corresponding help to display at the cursor.
    - Click the link under Page help to access the help panel for the administrative console panel. The help panel is the same help panel that displays when you click the **?** icon.
    - Expand the task help to view related tasks.

You can continue to access help information from the administrative console. Alternatively, you can access the help information from the WebSphere Application Server Information Center.

You can continue to access the WebSphere Application Server Information Center, the WebSphere Application Server product information, and the WebSphere Application Server technical information on developerWorks from the administrative console. Alternatively you can access the information from the IBM Web site.

Administrative console: Resources for learning

Use the following links to find relevant supplemental information about the IBM WebSphere Application Server 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.

View links to additional information:

Administration

- IBM WebSphere Application Server Redbooks

  This site contains a listing of all WebSphere Application Server Redbooks.
- IBM developerWorks WebSphere

  This site is the home of technical information for developers working with WebSphere products. You can download WebSphere software, take a fast path to developerWorks zones, such as VisualAge Java or WebSphere Application Server, learn about WebSphere products through a newcomers page, tutorials, technology previews, training, and Redbooks, get answers to questions about WebSphere products, and join the WebSphere community, where you can keep up with the latest developments and technical papers.
- WebSphere Application Server Support page

  Take advantage of the Web-based Support and Service resources from IBM to quickly find answers to your technical questions. You can easily access this extensive Web-based support through the IBM Software Support portal at URL `http://www-3.ibm.com/software/support/` and search by product category, or by product name. For example, if you are experiencing problems specific to WebSphere Application Server, click **WebSphere Application Server** in the product list. The WebSphere Application Server Support page appears.

Using scripting (wsadmin)

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

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

- Getting started with scripting Provides an introduction to WebSphere Application Server scripting and information about using the wsadmin tool. Topics include information about the scripting languages and the scripting objects, and instructions for starting the wsadmin tool.
- Deploying applications Provides instructions for deploying and uninstalling applications. For example, stand-alone Java archive files and Web archive files, the administrative console, remote Enterprise Archive (EAR) files, file transfer applications, and so on.
- Managing deployed applications Includes tasks that you perform after the application is deployed. For example, starting and stopping applications, checking status, modifying listener address ports, querying application state, configuring a shared library, and so on.
- Configuring servers Provides instructions for configuring servers, such as creating a server, modifying and restarting the server, configuring the Java virtual machine, disabling a component, disabling a service, and so on.
- Configuring connections to Web servers Includes topics such as regenerating the plug-in, creating new virtual host templates, modifying virtual hosts, and so on.
- Managing servers Includes tasks that you use to manage servers. For example, stopping nodes, starting and stopping servers, querying a server state, starting a listener port, and so on.
- Configuring security Includes security tasks, for example, enabling and disabling 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.
- Troubleshooting Provides information about how to troubleshoot using scripting. For example, tracing, thread dumps, profiles, and so on.
- 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.

Getting started with scripting

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 wsadmin tool supports a full range of product administrative activities.

The following figure illustrates the major components involved in a wsadmin scripting solution:

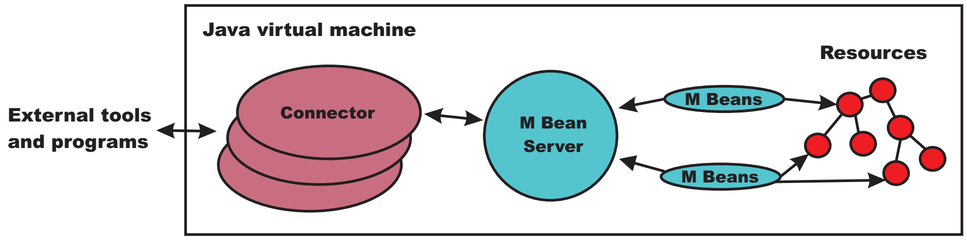


Figure 1: A WebSphere Application Server scripting solution

The wsadmin tool supports two scripting languages: Jacl and Jython. Five objects are available when you use scripts:

- **AdminControl**: Use to run operational commands.
- **AdminConfig**: Use to run configurational commands to create or modify WebSphere Application Server configurational elements.
- **AdminApp**: Use to administer applications.
- **AdminTask**: Use to run administrative commands.
- **Help**: Use to obtain general help.

The scripts use these objects to communicate with MBeans that run in WebSphere Application Server processes. MBeans are Java objects that represent Java Management Extensions (JMX) resources. JMX is an optional package addition to Java 2 Platform Standard Edition (J2SE). JMX is a technology that provides a simple and standard way to manage Java objects.

To perform a task using scripting, you must first perform the following steps:

1. Choose a scripting language. The wsadmin tool only supports Jacl and Jython scripting languages. Jacl is the language specified by default. If you want to use the Jython scripting language, use the -lang option or specify it in the wsadmin.properties file.
2. Start the wsadmin scripting client interactively, as an individual command, in a script, or in a profile.

Before you perform any task using scripting, make sure that you are familiar with the following concepts:

- Java Management Extensions (JMX)
- WebSphere Application Server configuration model
- wsadmin tool

- Jacl syntax or Jython syntax
- Scripting objects

Optionally, you can customize your scripting environment. For more information, see Scripting environment properties.

After you become familiar with the scripting concepts, choose a scripting language, and start the scripting client, you are ready to perform tasks using scripting.

Java Management Extensions (JMX)

Java Management Extensions (JMX) is a framework that provides a standard way of exposing Java resources, for example, application servers, to a system management infrastructure. Using the JMX framework, a provider can implement functions, such as listing the configuration settings, and editing the settings. This framework also includes a notification layer that management applications can use to monitor events such as the startup of an application server.

JMX key features

The key features of the WebSphere Application Server Version 6 implementation of JMX include:

- All processes that run the JMX agent.
- All run-time administration that is performed through JMX operations.
- Connectors that are used to connect a JMX agent to a remote JMX-enabled management application. The following connectors are supported:
  - SOAP JMX Connector
  - Remote Method Invocation over the Internet Inter-ORB Protocol (RMI-IIOP) JMX Connector
- Protocol adapters that provide a management view of the JMX agent through a given protocol. Management applications that connect to a protocol adapter are usually specific to a given protocol.
- The ability to query and update the configuration settings of a run-time object.
- The ability to load, initialize, change, and monitor application components and resources during run-time.

JMX architecture

The JMX architecture is structured into three layers:

- Instrumentation layer - Dictates how resources can be wrapped within special Java beans, called managed beans (MBeans).
- Agent layer - Consists of the MBean server and agents, which provide a management infrastructure. The services that are implemented include:
  - Monitoring
  - Event notification
  - Timers
- Management layer - Defines how external management applications can interact with the underlying layers in terms of protocols, APIs, and so on. This layer uses an implementation of the distributed services specification (JSR-077), which is not yet part of the Java 2 platform, Enterprise Edition (J2EE) specification.

The layered architecture of JMX is summarized in the following figure:

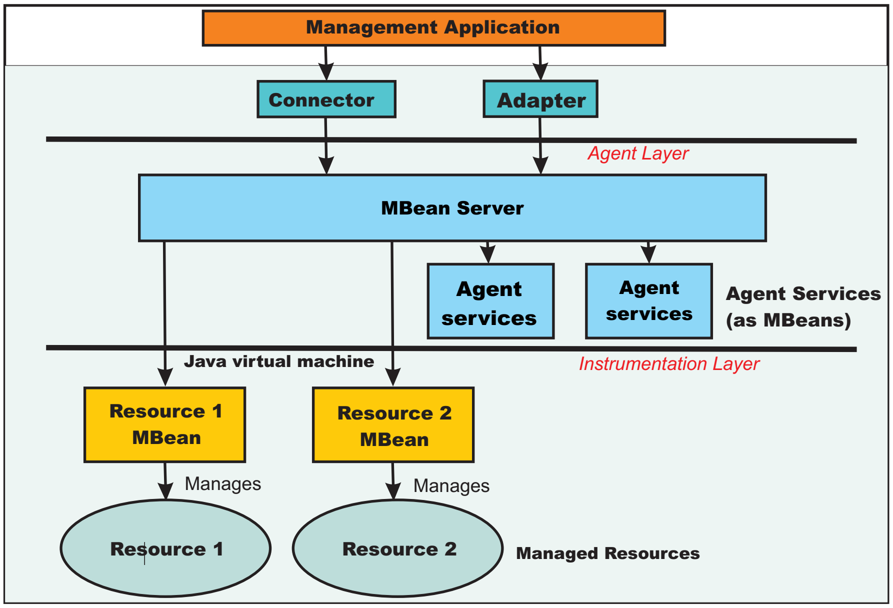


Figure 1: JMX architecture

JMX distributed administration

The following figure shows how the JMX architecture fits into the overall distributed administration topology of a Network Deployment environment:

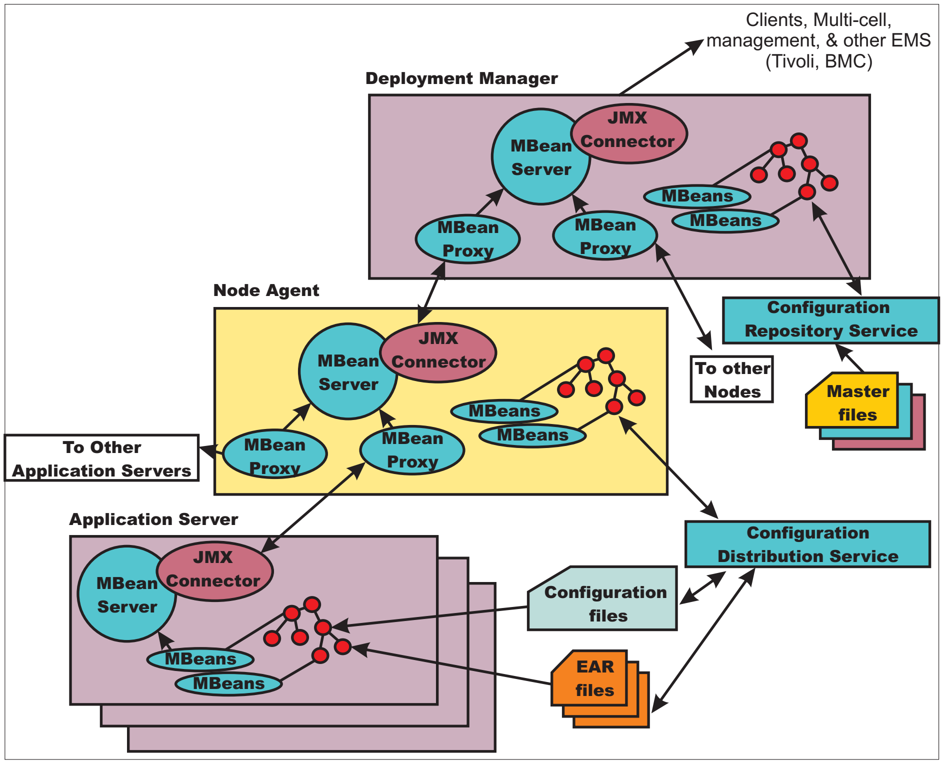


Figure 2: WebSphere Application Server distributed administration of JMX

The key points of this distributed administration architecture include:

- Internal MBeans that are local to the Java virtual machine (JVM) register with the local MBean server.
- External MBeans have a local proxy to their MBean server. The proxy registers with the local MBean server. Using the MBean proxy the local MBean server can pass the message to an external MBean server that is located on:
  - A node agent that has an MBean proxy for all the servers within its node. The MBean proxies for other nodes are not used.
  - The deployment manager has MBean proxies for all the node agents in the cell.

JMX Mbeans

WebSphere Application Server provides a number of MBeans, each of which have different functions and operations available. For example, an application server MBean can expose operations such as start and stop. An application MBean can expose operations such as install and uninstall. Some JMX usage scenarios that you can encounter include:

- External programs that are written to control the Network Deployment run time and its WebSphere resources by programmatically accessing the JMX API.
- Third-party applications that include custom JMX MBeans as part of the deployed code, supporting the JMX API management of application components and resources.

The following example illustrates how to obtain an MBean:

Using Jacl:

```
set am [$AdminControl queryNames type=ApplicationManager,process=server1,*]
```

Using Jacl:

```
am = AdminControl.queryNames('type=ApplicationManager,process=server1,*')
```

Each WebSphere Application Server runtime MBean may have attributes, operations, and notifications. The complete documentation for each MBean supplied with WebSphere Application Server is available in an html table that is installed in each copy of the WebSphere Application Server product. Under the main install directory for the product, there is a directory named web. And under that directory is another directory called mbeanDocs. In the mbeanDocs directory are several html files, one for each supplied with WebSphere Application Server. There is also an index.html file that ties all the individual MBean files together in a top-level navigation tree. Each MBean provides summary of its attributes, operations, and notifications.

JMX benefits

The use of JMX for management functions in WebSphere Application Server provides the following benefits:

- Enables the management of Java applications without significant investment.
- Relies on a core-managed object server that acts as a management agent.
- Java applications can embed a managed object server and make some of its functionality available as one or several MBeans that are registered with the object server.
- Provides a scalable management architecture.
- Every JMX agent service is an independent module that can be plugged into the management agent.
- The API is extensible, allowing new WebSphere Application Server and custom application features to be easily added and exposed through this management interface.
- Integrates existing management solutions.
- JMX smart agents are capable of being managed through HTML browsers or by various management protocols such as Web services, Java Message Service (JMS), and Simple Network Management Protocol (SNMP).

- Each process is self sufficient when it comes to the management of its resources. No central point of control exists. In principle, a JMX-enabled management client can be connected to any managed process and interact with the MBeans that are hosted by that process.
- JMX provides a single, flat, domain-wide approach to system management. Separate processes interact through MBean proxies that support a single management client to seamlessly navigate through a network of managed processes.
- Defines the interfaces that are necessary for management only.
- Provides a standard API for exposing application and administrative resources to management tools.

WebSphere Application Server configuration model

Configuration data is stored in several XML files. The server runtime reads these files when started and responds to the component settings stored there. The configuration data includes settings for the runtime itself, such as JVM options, thread pool sizes, container setting, and port numbers the server will use. Other configuration files define J2EE resources to which the server will connect to obtain data needed by the application logic. Security settings are stored in a separate document from the server and resource configuration. Applcation-specific configuration, such as deployment target lists, session configuration, and cache settings, are stored in files under the root directory of each application.

The configuration model for WebSphere Application Server is large and complex. When viewing the XML data in the various configuration files, you can discern relationship between the configuration objects. Understanding these relationships between configuration objects is essential when writing wsadmin scripts that perform configuration functions.

Full documentation of all of the WebSphere configuration objects is available in an html table that is installed in each copy of the WebSphere product. Under the main install directory for the product, there is a directory named web. And under that directory is another directory called configDocs. In the configDocs directory are several subdirectories, one for each configuration package in the model. There is also an index.html file that ties all the individual configuration packages together in a top-level navigation tree. Each configuration package lists all the supported configuration classes. Each configuration class lists all the supported properties. Properties with names that end with the special @ character imply that property is actually a reference to some other configuration object within the configuration data. Properties with names that end with an * imply that property is actaully a list of other configuration objects.

Jacl

Jacl is an alternate implementation of TCL, and is written entirely in Java code.

The wsadmin tool uses Jacl V1.3.1. The following information is a basic summary of the Jacl syntax:

Basic syntax:

The basic syntax for a Jacl command is the following:

```
Command arg1 arg2 arg3 ...
```

The command is either the name of a built-in command or a Jacl procedure. For example:

```
puts stdout {Hello, world!}
=> Hello, world!
```

In this example, the command is **puts** which takes two arguments, an I/O stream identifier and a string. The **puts** command writes the string to the I/O stream along with a trailing new line character. The arguments are interpreted by the command. In the example, `stdout` is used to identify the standard output stream. The use of `stdout` as a name is a convention employed by the **puts** command and the other I/O commands. `stderr` identifies the standard error output, and `stdin` identifies the standard input.

Variables

The **set** command assigns a value to a variable. This command takes two arguments: the name of the variable and the value. Variable names can be any length and are case sensitive. You do not have to declare Jacl variables before you use them. The interpreter will create the variable when it is first assigned a value. For example:

```
set a 5
=> 5
```

```
set b $a
=> 5
```

The second example assigns the value of variable `a` to variable `b`. The use of dollar sign ($) is indicates variable substitution. You can delete a variable with the **unset** command, for example:

```
unset varName1 varName2 ...
```

You can pass any number of variables to the **unset** command. The **unset** command will give error if a variable is not already defined. You can delete an entire array or just a single array element with the **unset** command. Using the **unset** command on an array is a easy way to clear out a big data structure. The existence of a variable can be tested with the **info exists** command. You may have to test for the existence of the variable because the `incr` parameter requires that a variable exist first, for example:

```
if ![info exists foobar] {set foobar 0} else {incr foobar}
```

Command substitution:

The second form of substitution is command substitution. A nested command is delimited by square brackets, [ ]. The Jacl interpreter evaluates everything between the brackets and evaluates it as a command. For example:

```
set len [string length foobar]
=> 6
```

In this example, the nested command is the following: `string length foobar`. The **string** command performs various operations on strings. In this case, the command asks for the length of the string `foobar`. If there are several cases of command substitution within a single command, the interpreter processes them from left bracket to right bracket. For example:

```
set number "1 2 3 4"
=> 1 2 3 4
```

```
set one [lindex $number 0]
=> 1
```

```
set end [lindex $number end]
=> 4
```

```
set another {123  456  789}
=> 123  456  789
```

```
set stringLen [string length [lindex  $another  1]]
=> 3
```

```
set listLen [llength [lindex $another 1]
=> 1
```

Math expressions:

The Jacl interpreter does not evaluate math expressions. Use the **expr** command to evaluate math expressions. The interpreter treats the **expr** command similar to other commands and leaves the expression parsing up the **expr** command implementation. The implementation of the **expr** command takes all arguments, concatenates them into a single string, and parses the string as a math expression. After the **expr** command computes the answer, it his formatted into a string and returned. For example:

```
expr 7.2 / 3
=> 2.4
```

Backslash substitution:

The final type of substitution done by the Jacl interpreter is backslash substitution. Use this to quote characters that have special meaning to the interpreter. For example, you can specify a literal dollar sign, brace, or bracket by quoting it with a backslash. If you are using lots of backslashes, instead you can group things with curly braces to turn off all interpretation of special characters. There are cases where backslashes are required. For example:

```
set dollar "This is a string \$contain dollar char"
=> This is a string $contain dollar char

set x $dollar
=> This is a string $contain dollar char

set  group {$ {} [] { [ } ]}
=> $ {} [] { [ } ]
```

You can also use backslashes to continue long commands on multiple lines. A new line without the backslash terminates a command. A backslashes that are the last character on a line convert into a space. For example:

```
set  totalLength  [expr  [string  length  "first string"] + \
[string  length  "second string"]]
=> 25
```

Grouping with braces and double quotes:

Use double quotes and curly braces to group words together. Quotes allow substitutions to occur in the group and curly braces prevent substitution. This rule applies to command, variable, and backslash substitutions. For example:

```
set s Hello
=> Hello

puts stdout "The length of $s is [string length $s]."
=> The length of Hello is 5.

puts  stdout  {The length of $s is [string  length  $s].}
=> The length of $s is [string length $s].
```

In the second example, the Jacl interpreter performs variable and command substitution on the second argument from the **puts** command. In the third command, substitutions are prevented so the string is printed as it is.

On distributed systems, special care must also be taken with path descriptions because the Jacl language uses the backslash character (\) as an escape character. To fix this, either replace each backslash with a forward slash, or use double backslashes in distributed path statements. For example: `C:/` or `C:\\`

Procedures and scope:

Jacl uses the **proc** command to define procedures. The basic syntax to define a procedure is the following:

```
proc  name  arglist  body
```

The first argument is the name of the procedure being defined. The name is case sensitive, and in fact it can contain any characters. Procedure names and variable names do not conflict with each other. The second argument is a list of parameters to the procedures. The third argument is a command, or more typically a group of commands that form the procedure body. Once defined, a Jacl procedure is used just like any of the built-in commands. For example:

```
proc divide {x y} {
set result [expr x/y]
puts $result
}
```

Inside the script, this is how to call devide procedure:

```
divide 20 5
```

And it will give the result like below:

```
4
```

It is not really necessary to use the variable c in this example. The procedure body could also written as:

```
return  [expr  sqrt($a  *  $a  +  $b  *  $b)]
```

The return command is optional in this example because the Jacl interpreter returns the value of the last command in the body as the value of the procedure. So, the procedure body could be reduced to:

```
expr  sqrt($a  *  $a  +  $b  *  $b)
```

The result of the procedure is the result returned by the last command in the body. The return command can be used to return a specific value.

There is a single, global scope for procedure names. You can define a procedure inside another procedure, but it is visible everywhere. There is a different name space for variables and procedures therefore you may have a procedure and a variable with the same name without a conflict. Each procedure has a local scope for variables. Variables introduced in the procedures only exist for the duration of the procedure call. After the procedure returns, those variables are undefined. If the same variable name exists in an outer scope, it is unaffected by the use of that variable name inside a procedure. Variables defined outside the procedure are not visible to a procedure, unless the global scope commands are used.

- **global** command - Global scope is the top level scope. This scope is outside of any procedure. You must make variables defined at the global scope accessible to the commands inside procedure by using the **global** command. The syntax for the **global** command is the following:

  ```
  global  varName1  varName2 ...
  ```

Comments

Use the pound character (#) to make comments.

Command line arguments

The Jacl shells pass the command line arguments to the script as the value of the `argv` variable. The number of command line arguments is given by `argc` variable. The name of the program, or script, is not part of `argv` nor is it counted by `argc`. Instead, it is put into the `argv0` variable. The `argv` variable is a list. Use the **lindex** command to extract items from the argument list, for example:

```
set  first  [lindex  $argv  0]
set  second  [lindex  $argv  1]
```

Strings and pattern matching

String are the basic data item in the Jacl language. There are multiple commands that you can use to manipulate strings. The general syntax of the **string** command is the following:

```
string operation stringvalue otherargs
```

The `operation` argument determines the action of the string. The second argument is a string value. There may be additional arguments depending on the operation.

The following table includes a summary of the **string** command:

Command	Description

string compare str1 str2	Compares strings lexicographically. Returns 0 if equal, -1 if str1 sorts before str2, else1.
string first str1 str2	Returns the index in str2 of the first occurrences of str1, or -1 if str1 is not found.
string index string index	Returns the character at the specified index.
string last str1 str2	Returns the index in str2 of the last occurence of str1, or -1 if str1 is not found.
string length string	Returns the number of character in string.
string match pattern str	Returns 1 if str matches the pattern, else 0.
string range str i j	Returns the range of characters in str from i to j
string tolower string	Returns string in lower case.
string toupper string	Returns string in upper case.
string trim string ?chars?	Trims the characters in chars from both ends of string. chars defaults to white space.
string trimleft string ?chars?	Trims the characters in chars from the beginning of string. chars defaults to white space.
string trimright string ?chars?	Trims the characters in chars from the end of string. chars defaults to white space.
string wordend str ix	Returns the index in str of the character after the word containing the character at index ix.
string wordstart str ix	Returns the index in str of the first character in the word containing the character at index ix.

The append command

The first argument of the **append** command is a variable name. It concatenates the remaining arguments onto the current value of the named variable. For example:

```
set  foo  z
=> z

append  foo a b c
=> zabc
```

The regexp command

The **regexp** command provides direct access to the regular expression matcher. The syntax is the following:

```
regexp  ?flags?  pattern  string  ?match  sub1  sub2  ...?
```

The return value is `1` if some part of the string matches the pattern. Otherwise, the return value will be `0`. The pattern does not have to match the whole string. If you need more control than this, you can anchor the pattern to the beginning of the string by starting the pattern with ^, or to the end of the string by ending the pattern with dollar sign, `$`. You can force the pattern to match the whole string by using both characters. For example:

```
set  text1  "This is the first string"
=> This is the first string

regexp  "first string" $text1
=> 1

regexp "second string"  $text1
=> 0
```

Jacl data structures

The basic data structure in the Jacl language is a string. There are two higher level data structures: lists and arrays. Lists are implemented as strings and the structure is defined by the syntax of the string. The syntax rules are the same as for commands. Commands are a particular instance of lists. Arrays are variables that have an index. The index is a string value so you can think of arrays as maps from one string (the index) to another string (the value of the array element).

Jacl lists

The lists of the Jacl language are strings with a special interpretation. In the Jacl language, a list has the same structure as a command. A list is a string with list elements separated by white space. You can use braces or quotes to group together words with white space into a single list element.

The following table includes commands that are related to lists:

Command	Description
list arg1 arg2	Creates a list out of all its arguments.
lindex list i	Returns the i'th element from list.
llength list	Returns the number of elements in list.
lrange list i j	Returns the i'th through j'th elements from list.
lappend listVar arg arg ...	Appends elements to the value of listVar
linsert list index arg arg ...	Inserts elements into list before the element at position index. Returns a new list.
lreplace list i j arg arg ...	Replaces elements i through j of list with the args. Return a new list.
lsearch mode list value	Returns the index of the element in list that matches the value according to the mode, which is -exact, -glob, or -regexp, -glob is the default. Return -1 if not found.
lsort switches list	Sorts elements of the list according to the switches: -ascii, -integer, -real, -increasing, -decreasing, -command command. Return a new list.
concat arg arg arg ...	Joins multiple lists together into one list.
join list joinString	Merges the elements of a list together by separating them with joinString.
split string splitChars	Splits a string up into list elements, using (and discarding) the characters in splitChars as boundaries between list elements.

Arrays

Arrays are the other primary data structure in the Jacl language. An array is a variable with a string-valued index, so you can think of an array as a mapping from strings to strings. Internally an array is implemented with a hash table. The cost of accessing each element is about the same. The index of an array is delimited by parentheses. The index can have any string value, and it can be the result of variable or command substitution. Array elements are defined with the **set** command, for example:

```
set arr(index) value
```

Substitute the dollar sign ($) to obtain the value of an array element, for example:

```
set foo $arr(index)
```

For example:

```
set fruit(best) kiwi
=> kiwi

set fruit(worst) peach
=> peach

set fruit(ok) banana
=> banana

array get fruit
=> ok banana worst peach best kiwi

array exists fruit
=> 1
```

The following table includes array commands:

Command	Description
array exists arr	Returns 1 if arr is an array variable.
array get arr	Returns a list that alternates between an index and the corresponding array value.
array names arr ?pattern?	Return the list of all indices defined for arr, or those that match the string match pattern.
array set arr list	Initializes the array arr from list, which should have the same form as the list returned by get.
array size arr	Returns the number of indices defined for arr.
array startsearch arr	Returns a search token for a search through arr.
array nextelement arr id	Returns the value of the next element in array in the search identified by the token id. Returns an empty string if no more elements remain in the search.
array anymore arr id	Returns 1 if more elements remain in the search.
array donesearch arr id	Ends the search identified by `id`.

Control flow commands

The following looping commands exist:

- `while`
- `foreach`
- `for`

The following are conditional commands:

- `if`
- `switch`

The following is an error handling command:

- `catch`

The following commands fine-tune control flow:

- `break`
- `continue`
- `return`
- `error`

If Then Else

The **if** command is the basic conditional command. If an expression is true, then execute one command body, otherwise execute another command body. The second command body (the else clause) is optional. The syntax of the command is the following:

```
if  boolean  then  body1  else  body2
```

The then and else keywords are optional. For example:

```
if {$x == 0} {
 puts stderr "Divide by zero!"
} else {
 set slope [expr $y/$x]
}
```

Switch

Use the **switch** command to branch to one of many commands depending on the value of an expression. You can choose based on pattern matching as well as simple comparisons. Any number of pattern-body pairs can be specified. If multiple patterns match, only the body of the first matching pattern is evaluated. The general form of the command is the following:

```
switch  flags  value  pat1  body1  pat2  body2  ...
```

You can also group all the pattern-body pairs into one argument:

```
switch  flags  value  {pat1  body1  pat2  body2  ...}
```

There are four possible flags that determines how value is matched.

- `-exact` Matches the value exactly to one of the patterns.
- `-glob` Uses glob-style pattern matching.
- `-regexp` Uses regular expression pattern matching.
- `--` No flag (or end of flags). Useful when value can begin with a dash (-).

For example:

```
switch  -exact  --  $value  {
 foo  {doFoo;  incr  count(foo)}
 bar {doBar;  return  $count(foo)}
 default   {incr  count(other)}
}
```

If the pattern that is associated with the last body is `default`, then the command body is started if no other patterns match. The `default` keyword only works on the last pattern-body pair. If you use the default pattern on an earlier body, it will be treated as a pattern to match the literal string default.

Foreach

The **foreach** command loops over a command body and assigns a loop variable to each of the values in a list. The syntax is the following:

```
foreach loopVar valueList commandBody
```

The first argument is the name of a variable. The command body runs one time for each element in the loop with the loop variable having successive values in the list. For example:

```
set numbers {1 3 5 7 11 13}
foreach num $numbers {
puts $num
}
```

The result from the previous example will be the following output, assuming that only one server exists in the environment. If there is more than one server, the information for all servers returns:

```
1
3
5
7
11
13
```

While

The **while** command takes two arguments; a test and a command body, for example:

```
while  booleanExpr  body
```

The **while** command repeatedly tests the boolean expression and runs the body if the expression is true (non-zero). For example:

```
set i 0
while {$i < 5} {
puts "i is $i"
incr i}
```

The result from the previous example will be like the following output, assuming that there is only one server. If there is more then one servers, it will print all of the servers:

```
i is 0
i is 1
i is 2
i is 3
i is 4
```

For

The **for** command is similar to the C language for statement. It takes four arguments, for example:

```
for initial test final body
```

The first argument is a command to initialize the loop. The second argument is a boolean expression which determines if the loop body will run. The third argument is a command that runs after the loop body: For example:

```
set numbers {1 3 5 7 11 13}
for {set i 0} {$i < [llength $numbers]} {incr i 1} {
puts "i is $i"
}
```

The result from previous example will be like the following output, assuming that there is only one server in the environment. If there is more then one server, it will print all of the servers:

```
i is 1
i is 3
i is 5
i is 7
i is 11
i is 13
```

Break and continue

You can control loop execution with the **break** and **continue** commands. The **break** command causes an immediate exit from a loop. The **continue** command causes the loop to continue with the next iteration.

Catch

An error will occur if you call a command with the wrong number of arguments or if the command detects some error condition particular to its implementation. An uncaught error prevents a script from running. Use the **catch** command trap such errors. The catch command takes two arguments, for example:

```
catch command ?resultVar?
```

The first argument is a command body. The second argument is the name of a variable that will contain the result of the command or an error message if the command raises an error. The **catch** command returns a value of zero if no error was caught or a value of one if the command catches an error. For example:

```
catch {expr 20 / 5} result
==> 0
puts $result
==> 4
catch {expr text / 5} result
==> 1
puts $result
==> syntax error in expression "text / 5"
```

Return

The **return** command is used to return from a procedure. It is needed if you want something to return before the end of the procedure body or if a contrast value needs to be returned.

Namespaces

Jacl keeps track of named entities such as variables, in namespaces. The wsadmin tool also adds entries to the global namespace for the scripting objects, such as, the AdminApp object

When you run a proc command, a local namespace is created and initialized with the names and the values of the parameters in the proc command. Variables are held in the local namespace while you run the proc command. When you stop the proc command, the local namespace is erased. The local namespace of the proc command implements the semantics of the automatic variables in languages such as C and Java.

While variables in the global namespace are visible to the top level code, they are not visible by default from within a proc command. To make them visible, declare the variables globally using the **global** command. For the variable names that you provide, the global command creates entries in the local namespace that point to the global namespace entries that actually define the variables.

If you use a scripting object provided by the wsadmin tool in a proc, you must declare it globally before you can use it, for example:

```
proc { ... } {
  global AdminConfig
  ... [$AdminConfig ...]
}
```

For more information about Jacl, see the Scripting: Resources for Learning article.

Jython

Jython is an alternate implementation of Python, and is written entirely in Java.

The wsadmin tool uses Jython V2.1. The following information is a basic summary of the Jython syntax:

Basic function

The function is either the name of a built-in function or a Jython function. For example:

```
print "Hello, World!"
=> Hello, World!

import sys
sys.stdout.write("Hello World!\n")
=> Hello World!
```

In the example, `print` identifies the standard output stream. You can use the built-in module by running import statements such as the previous example. The statement import runs the code in a module as part of the importing and returns the module object. `sys` is a built-in module of the Python language. In the Python language, modules are name spaces which are places where names are created. Names that reside in modules are called attributes. Modules correspond to files and the Python language creates a module object to contain all the names defined in the file. In other words, modules are name spaces.

Variable

To assign objects to names, the target of an assignment should be on the left side of an equal sign (=) and the object that you are assigning on the right side. The target on the left side can be a name or object component, and the object on the right side can be an arbitrary expression that computes an object. The following rules exist for assigning objects to names:

- Assignments create object references.
- Names are created when you assign them.
- You must assign a name before referencing it.

Variable name rules are similar to the rules for the C language, for example:

- An underscore character (_) or a letter plus any number of letters, digits or underscores

The following reserved words can not be used for variable names:

```
and     assert   break    class   continue
def     del      elif     else    except
exec    inally     for      from    global
if     importin   is       lambda
not    or      pass     print    raise
return  try      while
```

For example:

```
a  = 5
print a
=> 5

b =  a
print b
=> 5

text1, text2, text3, text4  = 'good', 'bad', 'pretty', 'ugly'
print text3
=> pretty
```

The second example assigns the value of variable a to variable b.

Types and operators

The following list contains a few of the built-in object types:

- Numbers. For example:

```
8, 3.133,  999L,  3+4j

num1 = int(10)
print num1
=> 10
```

- Strings. For example:

```
'name',  "name's", ''

print str(12345)
=> '12345'
```

- Lists. For example:

```
x = [1, [2,  'free'], 5]
y = [0, 1, 2, 3]
y.append(5)
print y
=> [0, 1, 2, 3, 5]

y.reverse()
print y
=> [5, 3, 2, 1, 0]

y.sort()
print y
=> [0, 1, 2, 3, 5]

print list("apple")
=> ['a', 'p', 'p', 'l', 'e']

print list((1,2,3,4,5))
=> [1, 2, 3, 4, 5]

test = "This is a test"
test.index("test")
=> 10

test.index('s')
=> 3
```

The following list contains a few of the operators:

- x or y

  y is evaluated only if x is false. For example:

```
print 0 or 1
=> 1
```

- x and y

  y is evaluated only if x is true. For example:

```
print 0 and 1
=>  0
```

- x +y , x - y

  Addition and concatenation, subtraction. For example:

```
print  6 + 7
=> 13

text1 = 'Something'
text2 = ' else'
print text1 + text2
=> Something else

list1 = [0, 1, 2, 3]
list2 = [4, 5, 6, 7]
print list1 + list2
```

```
=> [0, 1, 2, 3, 4, 5, 6, 7]

print  10 - 5
=> 5
```

- x * y, x / y, x % y

  Multiplication and repetition, division, remainder and format. For example:

  ```
  print 5 * 6
  => 30

  print 'test' * 3
  => test test test

  print 30 / 6
  => 5

  print 32 % 6
  => 2
  ```

- x[i], x[i:j], x(...)

  Indexing, slicing, function calls. For example:

  ```
  test = "This is a test"
  print  test[3]
  => s

  print test[3:10]
  => s is a

  print test[5:]
  => is a test

  print x[:-4]
  => This is a print len(test)
  => 14
  ```

- <, <=, >, >=, ==, <>, !=, is is not

  Comparison operators, identity tests. For example:

  ```
  l1 = [1, ('a', 3)]
  l2 = [1, ('a', 2)]
  l1 < l2, l1 == l2, l1 > l2, l1 <> l2, l1 != l2, l1 is l2, l1 is not l2
  => (0, 0, 1, 1, 1, 0, 1)
  ```

Backslash substitution

If a statement needs to span multiple lines, you can also add a black slash (\) at the end of the previous line to indicate you are continuing on the next line. For example:

```
text =  "This is a tests of a long lines" \
" continuing lines here."
print text
=> This is a tests of a long lines continuing lines here.
```

Functions and scope

Jython uses the `def` statement to define functions. Functions related statements include:

- `def`, `return`

  The `def` statement creates a function `boject` and assigns it to a name. The`return` statement sends a result object back to the caller. This is optional, and if it is not present, a function exits so that control flow falls off the end of the function body.

- `global`

The global statement declares module-level variables that are to be assigned. By default, all names assigned in a function are local to that function and exist only while the function runs. To assign a name in the enclosing module, list functions in a global statement.

The basic syntax to define a function is the following:

```
def name (arg1, arg2, ... ArgN):
statements
return value
```

where *name* is the name of the function being defined. It is followed by an open parenthesis, a close parenthesis and a colon. The arguments inside parenthesis include a list of parameters to the procedures. The next line after the colon is the body of the function. A group of commands that form the body of the function. After you define a Jython function, it is used just like any of the built-in functions. For example:

```
def  intersect(seq1, seq2):
  try:
     res = []
     for x in seq1:
     if x in seq2:
       res.append(x)
     return res
  except:
```

To call the function above, use the following command:

```
s1 = "SPAM"
s2 = "SCAM"
intersect(s1, s2)
=> [S, A, M]

intersect([1,2,3], (1.4))
=> [1]
```

Comments

Make comments in the Jython language with the pound character (#).

Command line arguments

The Jython shells pass the command line arguments to the script as the value of the sys.argv. The name of the program, or script, is not part of sys.argv. sys.argv is an array, so you use the index command to extract items from the argument list, for example:

```
import sys
first  =  sys.argv[0]
second  = sys.argv[1]
arglen = len(sys.argv)
```

Basic statements

There are two looping statements: while and for. The conditional statement is if. The error handling statement is try. Finally, there are some statements to fine-tune control flow: break, continue and pass. The following is a list of syntax rules in Python:

- Statements run one after another until you say otherwise. Statements normally end at the end of the line they appear on. When statements are too long to fit on a single line you can also add a back sash (\) at the end of the prior line to indicate you are continuing on the next line.
- Block and statement boundaries are detected automatically. There are no braces, or begin or end delimiter, around blocks of code. Instead, the Python language uses the indentation of statements under a header in order to group the statements in a nested block. Block boundaries are detected by line indentation. All statements indented the same distance to the right belong to the same block of code until that block is ended by a line less indented.

- Compound statements = header; ':', indented statements. All compound statements in the Python language follow the same pattern: a header line terminated with a colon, followed by one or more nested statements indented under the header. The indented statements are called a block.
- Spaces and comments are usually ignored. Spaces inside statements and expressions are almost always ignored (except in string constants and indentation), so are comments.

If

The `if` statement selects actions to perform. The `if` statement may contain other statements, including other `if` statements. The `if` statement can be followed by one or more optional `elif` statements and ends with an optional `else` block.

The general format of an if looks like the following:

```
if test1
 statements1
elif test2
 statements2
else test3
 statements3
```

For example:

```
weather = 'sunny'
if weather == 'sunny':
 print "Nice weather"
elif weather == 'raining':
 print "Bad weather"
else:
 print "Uncertain, don't plan anything"
```

While

The `while` statement consists of a header line with a test expression, a body of one or more indented statements, and an optional `else` statement that runs if control exits the loop without running into a break statement. The `while` statement repeatedly executes a block of indented statements as long as a test at the top keeps evaluating a true value. The general format of an while looks like the following:

```
while test1
 statements1
else
 statements2
```

For example:

```
a = 0; b = 10
while a < b:
 print a
 a = a + 1
```

For

The `for` statement begins with a header line that specifies an assignment target or targets, along with an object you want to step through. The header is followed by a block of indented statements which you want to repeat.

The general format of an while looks like the following:

```
for target in object:
 statements
else:
 statements
```

It assigns items in the sequence object to the target, one by one, and runs the loop body for each. The loop body typically uses the assignment target to refer to the current item in the sequence as if it were a cursor stepping through the sequence. For example:

```
sum = 0
for x in [1, 2, 3, 4]:
  sum = sum + x
```

Break, continue, and pass

You can control running a loop the `break`, `continue` and `pass` statements. The `break` statement jumps out of the closest enclosing loop (past the entire loop statement). The `continue` statements jumps to the top of the closest enclosing loop (to the header line of the loop), and the `pass` statement is an empty statement placeholder.

Try

A statement will raise an error if it is called with the wrong number of arguments, or if it detects some error condition particular to its implementation. An uncaught error aborts execution of a script. The try statement is used to trap such errors. Python try statements come in two flavors, one that handles exceptions and one that executes finalization code whether exceptions occur or not. The `try`, `except`, `else` statement starts with a try header line followed by a block of indented statements, then one or more optional except clauses that name exceptions to be caught, and an optional `else` clause at the end. The `try`, `finally` statements starts with a try header line followed by a block of indented statements, then finally clause that always runs on the way out whether an exception occurred while the try block was running or not.

The general format of the try, except, else function looks like the following:

```
try:
 statements
except name:
 statements
except name, data:
 statements
else
 statements
```

For example:

```
try:
   myfunction()
except:
   import sys
   print 'uncaught exception', sys.exc_type, sys.exc_value

try:
   myfilereader()
except EOFError:
   break
else:
   process next line here
```

The general format of a try and finally looks like the following:

```
try:
 statements
finally:
 statements
```

For example:

```
def divide(x, y):
   return x / y
```

```
def tester(y):
   try:
     print divide(8, y)
   finally:
   print 'on the way out...'
```

For more information about the Jython language, see the Scripting: Resources for Learning article.

Scripting objects

The wsadmin tool operates on configurations and running objects through the following set of management objects: AdminConfig, AdminControl, AdminApp, AdminTask, and Help. Each of these objects has commands that you can use to perform administrative tasks. To use the scripting objects, specify the scripting object, a command, and command parameters. For example:

Using Jacl:

```
$AdminConfig attributes ApplicationServer
```

Using Jython:

```
print AdminConfig.attributes('ApplicationServer')
```

where `AdminConfig` is the scripting object, `attributes` is the command, and `ApplicationServer` is the command parameter.

To find out more specific information about each of the scripting objects, including command and command parameter information, see AdminConfig, AdminApp, AdminControl, AdminTask, or Help.

WebSphere Application Server system management separates administrative functions into two categories: functions that work with the configuration of WebSphere Application Server installations, and functions that work with the currently running objects in WebSphere Application Server installations.

Scripts work with both categories of objects. For example, an application server is divided into two distinct entities. One entity represents the configuration of the server that resides persistently in a repository on permanent storage. You can create, query, change, or remove this configuration without starting an application server process. The **AdminConfig object**, the **AdminTask object**, and the **AdminApp object** handle configuration functionality. You can invoke configuration functions with or without being connected to a server.

The second entity represents the running instance of an application server by a *Java Management Extensions (JMX) MBean*. This instance can have attributes that you can interrogate and change, and operations that you can invoke. These operational actions taken against a running application server do not have an effect on the persistent configuration of the server. The attributes that support manipulation from an MBean differ from the attributes that the corresponding configuration supports. The configuration can include many attributes that you cannot query or set from the running object. The WebSphere Application Server scripting support provides functions to locate configuration objects, and running objects. Objects in the configuration do not always represent objects that are currently running. The **AdminControl object** manages running objects.

You can use the **Help object** to obtain information about the AdminConfig, AdminApp, AdminControl, and AdminTask objects, to obtain interface information about running MBeans, and to obtain help for warnings and error messages.

Help object for scripted administration: The Help object provides general help, online information about running MBeans, and help on messages.

Use the Help object to obtain general help for the other objects supplied by the wsadmin tool for scripting: the AdminApp, AdminConfig, AdminTask, and AdminControl objects. For example, using Jacl, `$Help AdminApp` or using Jython, `Help.Adminapp()`, provides information about the AdminApp object and the available commands.

The Help object also to provides interface information about MBeans running in the system. The commands that you use to get online information about the running MBeans include: **all**, **attributes**, **classname**, **constructors**, **description**, **notification**, **operations**.

You can also use the Help object to obtain information about messages using the **message** command. The **message** command provides aid to understand the cause of a warning or error message and find a solution for the problem. For example, you receive a `WASX7115E` error when running the AdminApp **install** command to install an application, use the following example:

Using Jacl:

```
$Help message WASX7115E
```

Using Jython:

```
print Help.message('WASX7115E')
```

Example output:

```
Explanation: wsadmin failed to read an ear file when
preparing to copy it to a temporary location for AdminApp
processing.  User action: Examine the wsadmin.traceout
log file to determine the problem; there may be file permission problems.
```

The user action specifies the recommended action to correct the problem. It is important to understand that in some cases the user action may not be able to provide corrective actions to cover all the possible causes of an error. It is an aid to provide you with information to troubleshoot a problem.

To see a list of all available commands for the Help object, see the Commands for the Help object article or you can also use the **Help** command, for example:

Using Jacl:

```
$Help help
```

Using Jython:

```
print Help.help()
```

AdminApp object for scripted administration: Use the AdminApp object to manage applications. This object communicates with the WebSphere Application Server run time application management object to make application inquires and changes, for example:

- Installing and uninstalling applications
- Listing applications
- Editing applications or modules

Since applications are part of configuration data, any changes that you make to an application is kept in the configuration session, similar to other configuration data. Be sure to save your application changes so that the data transfers from the configuration session to the master repository.

With the application already installed, the AdminApp object can update application metadata, map virtual hosts to Web modules, and map servers to modules. You must perform any other changes, such as, specifying a library for the application to use or setting session management configuration properties, using the AdminConfig object.

You can run the commands for the AdminApp object in local mode. If a server is running, it is not recommended that you run the scripting client in local mode because any configuration changes that are made in local mode will not be reflected in the running server configuration and vice versa. If you save a conflicting configuration, you could corrupt the configuration. In a deployment manager environment, configuration updates are available only if a scripting client is connected to a deployment manager. When connected to a node agent or a managed application server, you will not be able to update the configuration because the configuration for these server processes are copies of the master configuration which resides in the deployment manager. The copies are created on a node machine when a configuration synchronization occurs between the deployment manager and the node agent. Make configuration changes to the server processes by connecting a scripting client to a deployment manager. For this reason, to change a configuration, do not run a scripting client in local mode on a node machine. It is not a supported configuration.

To see a list of all available commands for the AdminApp object, see the Commands for the AdminApp object article or you can also use the **Help** command, for example:

Using Jacl:

```
$AdminApp help
```

Using Jython:

```
print AdminApp.help()
```

Listing applications with the wsadmin tool:

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

Query the configuration and create a list of installed applications, for example:

- Using Jacl:

  ```
  $AdminApp list
  ```

- Using Jython:

  ```
  AdminApp.list()
  ```

where:

`$`	is a Jacl operator for substituting a variable name with its value
`AdminApp`	is an object allowing application objects management
`list`	is an AdminApp command

Example output:

```
DefaultApplication
SampleApp
app1serv2
```

Editing application configurations with the wsadmin tool:

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

1. Edit the entire application or a single application module. Use one of the following commands:
   - The following command uses the installed application and the command option information to edit the application:
     – Using Jacl:

```
$AdminApp edit appname {options}
```

- Using Jython list:

```
AdminApp.edit('appname', ['options'])
```

- Using Jython string:

```
AdminApp.edit('appname', '[options]')
```

where:

`$`	is a Jacl operator for substituting a variable name with its value
`AdminApp`	is an object allowing application objects management
`edit`	is an AdminApp command
appname	is the name of application or application module to edit. For the application module name, use the module name returned from **listModules** command as the value.
`{options}`	is a list of edit options and tasks similar to the ones for the install command

- The following command changes the application information by prompting you through a series of editing tasks:
  - Using Jacl:

```
$AdminApp editInteractive appname
```

  - Using Jython:

```
AdminApp.editInteractive('appname')
```

where:

`$`	is a Jacl operator for substituting a variable name with its value
`AdminApp`	is an object allowing application objects management
`editInteractive`	is an AdminApp command
appname	is the name of application or application module to edit. For the application module name, use the module name returned from **listModules** command as the value.

2. Save the configuration changes. See the “Saving configuration changes with the wsadmin tool” on page 232 article for more information.
3. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 article for more information.

AdminControl object for scripted administration: The AdminControl scripting object is used for operational control. It communicates with MBeans that represent live objects running a WebSphere server process. It includes commands to query existing running objects and their attributes and invoke operation on the running objects. In addition to the operational commands, the AdminControl object supports commands to query information on the connected server, convenient commands for client tracing, reconnecting to a server, and start and stop server for network deployment environment.

Many of the operational commands have two sets of signatures so that they can either invoke using string based parameters or using Java Management Extension (JMX) objects as parameters. Depending on the server process to which a scripting client is connected, the number and type of MBeans available varies. If a scripting client is connected to a deployment manager, then all MBeans in all server processes are visible. If a scripting client is connected to a node agent, all MBeans in all server processes on that node are accessible. When connected to an application server, only MBeans running in that application server are visible.

The following steps provide a general method to manage the cycle of an application:

- Install the application.
- Edit the application.
- Update the application.
- Uninstall the application.

To see a list of all available commands for the AdminControl object, see the Commands for the AdminControl object article or you can also use the **Help** command, for example:

Using Jacl:

```
$AdminControl help
```

Using Jython:

```
print AdminControl.help()
```

ObjectName, Attribute, and AttributeList classes:

WebSphere Application Server scripting commands use the underlying Java Management Extensions (JMX) classes, ObjectName, Attribute, and AttributeList, to manipulate object names, attributes and attribute lists respectively.

The WebSphere Application Server ObjectName class uniquely identifies running objects. The ObjectName class consists of the following elements:

- The domain name `WebSphere`.
- Several key properties, for example:
  - **`type`** - Indicates the type of object that is accessible through the MBean, for example, ApplicationServer, and EJBContainer.
  - **`name`** - Represents the display name of the particular object, for example, MyServer.
  - **`node`** - Represents the name of the node on which the object runs.
  - **`process`** - Represents the name of the server process in which the object runs.
  - **`mbeanIdentifier`** - Correlates the MBean instance with corresponding configuration data.

When ObjectName classes are represented by strings, they have the following pattern:

```
[domainName]:property=value[,property=value]*
```

For example, you can specify `WebSphere:name="My Server",type=ApplicationServer,node=n1,*` to specify an application server named `My Server` on node n1. (The asterisk (*) is a wildcard character, used so that you do not have to specify the entire set of key properties.) The AdminControl commands that take strings as parameters expect strings that look like this example when specifying running objects (MBeans). You can obtain the object name for a running object with the **getObjectName** command.

Attributes of these objects consist of a name and a value. You can extract the name and value with the **getName** and the **getValue** methods that are available in the javax.management.Attribute class. You can also extract a list of attributes.

Example: Collecting arguments for the AdminControl object: Verify that the arguments parameter is a single string. Each individual argument in the string can contain spaces. Collect each argument that contains spaces in some way.

- An example of how to obtain an MBean follows:

  Using Jacl:

  ```
  set am [$AdminControl queryNames type=ApplicationManager,process=server1,*]
  ```

  Using Jython:

  ```
  am = AdminControl.queryNames('type=ApplicationManager,process=server1,*')
  ```
- Multiple ways exist to collect arguments that contain spaces. Choose one of the following alternatives:

Using Jacl:

- – `$AdminControl invoke $am startApplication {"JavaMail Sample"}`
- – `$AdminControl invoke $am startApplication {{JavaMail Sample}}`
- – `$AdminControl invoke $am startApplication "\"JavaMail Sample\""`

Using Jython:

- – `AdminControl.invoke(am, 'startApplication', '[JavaMail Sample]')`
- – `AdminControl.invoke(am, 'startApplication', '\"JavaMail Sample\"')`

Example: Identifying running objects: In the WebSphere Application Server, MBeans represent running objects. You can interrogate the MBean server to see the objects it contains. Use the AdminControl object to interact with running MBeans.

- Use the **queryNames** command to see running MBean objects. For example:

  Using Jacl:

  ```
  $AdminControl queryNames *
  ```

  Using Jython:

  ```
  print AdminControl.queryNames('*')
  ```

  This command returns a list of all MBean types. Depending on the server to which your scripting client attaches, this list can contain MBeans that run on different servers:
  - – If the client attaches to a stand-alone WebSphere Application Server, the list contains MBeans that run on that server.
  - – If the client attaches to a node agent, the list contains MBeans that run in the node agent and MBeans that run on all application servers on that node.
  - – If the client attaches to a deployment manager, the list contains MBeans that run in the deployment manager, all of the node agents communicating with that deployment manager, and all application servers on the nodes served by those node agents.
- The list that the `queryNames` command returns is a string representation of JMX `ObjectName` objects. For example:

  ```
  WebSphere:cell=MyCell,name=TraceService,mbeanIdentifier=TraceService,type=TraceService,node=MyNode,process=server1
  ```

  This example represents a `TraceServer` object that runs in *server1* on *MyNode*.
- The single queryNames argument represents the `ObjectName` object for which you are searching. The asterisk ("*") in the example means return all objects, but it is possible to be more specific. As shown in the example, `ObjectName` has two parts: a domain, and a list of key properties. For MBeans created by the WebSphere Application Server, the domain is WebSphere. If you do not specify a domain when you invoke queryNames, the scripting client assumes the domain is WebSphere. This means that the first example query above is equivalent to:

  Using Jacl:

  ```
  $AdminControl queryNames WebSphere:*
  ```

  Using Jython:

  ```
  AdminControl.queryNames('WebSphere:*')
  ```
- WebSphere Application Server includes the following key properties for the `ObjectName` object:
  - – name
  - – type
  - – cell
  - – node
  - – process
  - – mbeanIdentifier

  These key properties are common. There are other key properties that exist. You can use any of these key properties to narrow the scope of the **queryNames** command. For example:

  Using Jacl:

  ```
  $AdminControl queryNames WebSphere:type=Server,node=myNode,*
  ```

  Using Jython:

```
AdminControl.queryNames('WebSphere:type=Server,node=myNode,*')
```

This example returns a list of all MBeans that represent server objects running the node *myNode*. The, * at the end of the ObjectName object is a JMX wildcard designation. For example, if you enter the following:

Using Jacl:

```
$AdminControl queryNames WebSphere:type=Server,node=myNode
```

Using Jython:

```
print AdminControl.queryNames('WebSphere:type=Server,node=myNode')
```

you get an empty list back because the argument to queryNames is not a wildcard. There is no Server MBean running that has exactly these key properties and no others.

- If you want to see all the MBeans representing applications running on a particular node, invoke the following example:

  Using Jacl:

  ```
  $AdminControl queryNames WebSphere:type=Application,node=myNode,*
  ```

  Using Jython:

  ```
  print AdminControl.queryNames('WebSphere:type=Application,node=myNode,*')
  ```

Specifying running objects using the wsadmin tool:

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

Perform the following steps to specify running objects:

1. Obtain the configuration ID with one of the following ways:
   - Obtain the object name with the **completeObjectName** command, for example:
     – Using Jacl:

     ```
     set var [$AdminControl completeObjectName template]
     ```

     – Using Jython:

     ```
     var = AdminControl.completeObjectName(template)
     ```

     where:

set	is a Jacl command
var	is a variable name
$	is a Jacl operator for substituting a variable name with its value
AdminControl	is an object that enables the manipulation of MBeans running in a WebSphere server process
completeObjectName	is an AdminControl command
template	is a string containing a segment of the object name to be matched. The template has the same format as an object name with the following pattern: [domainName]:property=value[,property=value]*. See Object name, Attribute, Attribute list for more information.

     If there are several MBeans that match the template, the **completeObjectName** command only retuns the first match. The matching MBean object name is then assigned to a variable.

     To look for *server1* MBean in *mynode*, use the following example:

     – Using Jacl:

     ```
     set server1 [$AdminControl completeObjectName node=mynode,type=Server,name=server1,*]
     ```

     – Using Jython:

```
server1 = AdminControl.completeObjectName('node=mynode,type=Server,name=server1,*')
```

- Obtain the object name with the **queryNames** command, for example:
  - Using Jacl:

    ```
    set var [$AdminControl queryNames template]
    ```
  - Using Jython:

    ```
    var = AdminControl.queryNames(template)
    ```

  where:

set	is a Jacl command
var	is a variable name
$	is a Jacl operator for substituting a variable name with its value
AdminControl	is an object that enables the manipulation of MBeans running in a WebSphere Application server process.
queryNames	is an AdminControl command
template	is a string containing a segment of the object name to be matched. The template has the same format as an object name with the following pattern: [domainName]:property=value[,property=value]*

2. If there are more than one running objects returned from the **queryNames** command, the objects are returned in a list syntax. One simple way to retrieve a single element from the list is to use the **lindex** command in Jacl and **split** command in Jython. The following example retrieves the first running object from the server list:
   - Using Jacl:

     ```
     set allServers [$AdminControl queryNames type=Server,*]
     set aServer [lindex $allServers 0]
     ```
   - Using Jython:

     ```
     allServers = AdminControl.queryNames('type=Server,*')

     # get line separator
     import  java
     lineSeparator = java.lang.System.getProperty('line.separator')

     aServer = allServers.split(lineSeparator)[0]
     ```

   For other ways to manipulate the list and then perform pattern matching to look for a specified configuration object, refer to the Jacl syntax.

You can now use the running object in with other AdminControl commands that require an object name as a parameter.

Identifying attributes and operations for running objects with the wsadmin tool:

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

Use the Help object **attributes** or **operations** commands to find information on a running MBean in the server.

1. Specify a running object.
2. Use the **attributes** command to display the attributes of the running object:
   - Using Jacl:

     ```
     $Help attributes MBeanObjectName
     ```
   - Using Jython:

```
Help.attributes(MBeanObjectName)
```

where:

`$`	is a Jacl operator for substituting a variable name with its value
`Help`	is the object that provides general help and information for running MBeans in the connected server process
`attributes`	is a Help command
`MBeanObjectName`	is the string representation of the MBean object name obtained in step 2

3. Use the **operations** command to find out the operations supported by the MBean:
   - Using Jacl:

     ```
     $Help operations MBeanObjectname
     ```

     or

     ```
     $Help operations MBeanObjectname operationName
     ```

   - Using Jython:

     ```
     Help.operations(MBeanObjectname)
     ```

     or

     ```
     Help.operations(MBeanObjectname, operationName)
     ```

   where:

`$`	is a Jacl operator for substituting a variable name with its value
`Help`	is the object that provides general help and information for running MBeans in the connected server process
`operations`	is a Help command
`MBeanObjectname`	is the string representation of the MBean object name obtained in step number 2
`operationName`	(optional) is the specified operation for which you want to obtain detailed information

If you do not provide the `operationName`, all operations supported by the MBean return with the signature for each operation. If you specify `operationName`, only the operation that you specify returns and it contains details which include the input parameters and the return value. To display the operations for the server MBean, use the following example:

- Using Jacl:

  ```
  set server [$AdminControl completeObjectName type=Server,name=server1,*]
  $Help operations $server
  ```

- Using Jython:

  ```
  server = AdminControl.completeObjectName('type=Server,name=server1,*')
  print Help.operations(server)
  ```

To display detailed information about the stop operation, use the following example:

- Using Jacl:

  ```
  $Help operations $server stop
  ```

- Using Jython:

  ```
  print Help.operations(server, 'stop')
  ```

Performing operations on running objects using the wsadmin tool:

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

Perform the following steps to perform operations on running objects:

1. Obtain the object name of the running object. For example:
   - Using Jacl:
     ```
     $AdminControl completeObjectName name
     ```
   - Using Jython:
     ```
     AdminControl.completeObjectName(name)
     ```

   where:

$	is a Jacl operator for substituting a variable name with its value
AdminControl	is an object that enables the manipulation of MBeans running in a WebSphere server process
completeObjectName	is an AdminControl command
name	is a fragment of the object name. It is used to find the matching object name. For example: type=Server,name=serv1,*. It can be any valid combination of domain and key properties. For example, type, name, cell, node, process, etc.

2. Set the s1 variable to the running object, for example:
   - Using Jacl:
     ```
     set s1 [$AdminControl completeObjectName type=Server,name=server1,*]
     ```
   - Using Jython:
     ```
     s1 = AdminControl.completeObjectName('type=Server,name=server1,*')
     ```

   where:

set	is a Jacl command
s1	is a variable name
$	is a Jacl operator for substituting a variable name with its value
AdminControl	is an object that enables the manipulation of MBeans running in a WebSphere server process
completeObjectName	is an AdminControl command
type	is the object name property key
Server	is the name of the object
name	is the object name property key
server1	is the name of the server where the operation will be invoked

3. Invoke the operation. For example:
   - Using Jacl:
     ```
     $AdminControl invoke $s1 stop
     ```
   - Using Jython:
     ```
     AdminControl.invoke(s1, 'stop')
     ```

where:

$	is a Jacl operator for substituting a variable name with its value
AdminControl	is an object that enables the manipulation of MBeans running in a WebSphere server process
invoke	is an AdminControl command
s1	is the ID of the server specified in step number 3
stop	is an operation to be invoked on the server

The following example is for operations that require parameters:

- Using Jacl:

```
set traceServ [$AdminControl completeObjectName type=TraceService,process=server1,*]
$AdminControl invoke $traceServ appendTraceString "com.ibm.ws.management.*=all=enabled"
```

- Using Jython:

```
traceServ = AdminControl.completeObjectName('type=TraceService,process=server1,*')
AdminControl.invoke(traceServ, 'appendTraceString',  "com.ibm.ws.management.*=all=enabled")
```

Modifying attributes on running objects with the wsadmin tool:

Before starting this task, the wsadmin tool must be running. See the "Starting the wsadmin scripting client" on page 250 article for more information.

Perform the following steps to modify attributes on running objects:

1. Obtain the name of the running object, for example:
   - Using Jacl:

     ```
     $AdminControl completeObjectName name
     ```

   - Using Jython:

     ```
     AdminControl.completeObjectName(name)
     ```

   where:

$	is a Jacl operator for substituting a variable name with its value
AdminControl	is an object that enables the manipulation of MBeans running in a WebSphere server process
completeObjectName	is an AdminControl command
name	is a fragment of the object name. It is used to find the matching object name. For example: type=TraceService,node=mynode,*. It can be any valid combination of domain and key properties. For example, type, name, cell, node, process, etc.

2. Set the ts1 variable to the running object, for example:
   - Using Jacl:

     ```
     set ts1 [$AdminControl completeObjectName name]
     ```

   - Using Jython:

     ```
     ts1 = AdminControl.completeObjectName(name)
     ```

   where:

set	is a Jacl command

ts1	is a variable name
$	is a Jacl operator for substituting a variable name with its value
AdminControl	is an object that enables the manipulation of MBeans running in a WebSphere server process
completeObjectName	is an AdminControl command
name	is a fragment of the object name. It is used to find the matching object name. For example: `type=TraceService,node=mynode,*`. It can be any valid combination of domain and key properties. For example, type, name, cell, node, process, etc.

3. Modify the running object, for example:
   - Using Jacl:

     ```
     $AdminControl setAttribute $ts1 ringBufferSize 10
     ```

   - Using Jython:

     ```
     AdminControl.setAttribute(ts1, 'ringBufferSize', 10)
     ```

   where:

$	is a Jacl operator for substituting a variable name with its value
AdminControl	is an object that enables the manipulation of MBeans running in a WebSphere server process
setAttribute	is an AdminControl command
ts1	evaluates to the ID of the server specified in step number 3
ringBufferSize	is an attribute of modify objects
10	is the value of the ringBufferSize attribute

   You can also modify multiple attribute name and value pairs, for example:
   - Using Jacl:

     ```
     set ts1 [$AdminControl completeObjectName type=TraceService,process=server1,*]
     $AdminControl setAttributes $ts1 {{ringBufferSize 10} {traceSpecification com.ibm.*=all=disabled}}
     ```

   - Using Jython list:

     ```
     ts1 = AdminControl.completeObjectName('type=TraceService,process=server1,*')
     AdminControl.setAttributes(ts1, [['ringBufferSize', 10], ['traceSpecification',  'com.ibm.*=all=disabled']])
     ```

   - Using Jython string:

     ```
     ts1 =AdminControl.completeObjectName('type=TraceService,process=server1,*')
     AdminControl.setAttributes(ts1, '[[ringBufferSize 10] [traceSpecification  com.ibm.*=all=disabled]]')
     ```

   The new attribute values are returned to the command line.

Synchronizing nodes with the wsadmin tool:

This article only applies to network deployment installations. A node synchronization is necessary in order to propagate configuration changes to the affected node or nodes. By default this occurs periodically, as long as the node can communicate with the deployment manager. It is possible to cause this to happen explicitly by performing the following steps:

1. Set the variable for node synchronize.
   - Using Jacl:

     ```
     set Sync1 [$AdminControl completeObjectName type=NodeSync,node=myNodeName,*]
     ```

- Using Jython:

  ```
  Sync1 = AdminControl.completeObjectName('type=NodeSync,node=myNodeName,*')
  ```

  where:

`set`	is a Jacl command
`Sync1`	is a variable name
`$`	is a Jacl operator for substituting a variable name with its value
`AdminControl`	is an object that enables the manipulation of MBeans running in a WebSphere server process
`completeObjectName`	is an AdminControl command
`type=NodeSync,node=`*`myNodeName`*	is a fragment of the object name whose complete name is returned by this command. It is used to find the matching object name which is, in this case, the SyncNode object for the node *myNodeName*, where *myNodeName* is the name of the node that you use to synchronize configuration changes. For example: `type=Server, name=serv1`. It can be any valid combination of domain and key properties. For example, type, name, cell, node, process, etc.

  Example output:

  ```
  WebSphere:platform=common,cell=myNetwork,version=5.0,name=node
  Sync,mbeanIdentifier=nodeSync,type=NodeSync,node=myBaseNode,
  process=nodeagent
  ```

2. Synchronize by issuing the following command:
   - Using Jacl:

     ```
     $AdminControl invoke $Sync1 sync
     ```

   - Using Jython:

     ```
     AdminControl.invoke(Sync1, 'sync')
     ```

   where:

`$`	is a Jacl operator for substituting a variable name with its value
`AdminControl`	is an object that enables the manipulation of MBeans running in a WebSphere server process
`invoke`	is an AdminControl command
`Sync1`	evaluates to the ID of the server specified in step number 7
`sync`	is an attribute of modify objects

   Example output:

   ```
   true
   ```

   You will receive an output value of `true` if the synchronization completes.

When the synchronization is complete, the files created in the `c:/WebSphere/DeploymentManager/config` directory now exists on the *`mynode`* node in the `c:/WebSphere/AppServer/config` directory.

AdminConfig object for scripted administration: Use the AdminConfig object to manage the configuration information that is stored in the repository. This object communicates with the WebSphere

Application Server configuration service component to make configuration inquires and changes. You can use it to query existing configuration objects, create configuration objects, modify existing objects, remove configuration objects, and obtain help.

Updates to the configuration through a scripting client are kept in a private temporary area called a workspace and are not copied to the master configuration repository until you run a **save** command. The workspace is a temporary repository of configuration information that administrative clients including the administrative console use. The workspace is kept in the `wstemp` subdirectory of your WebSphere Application Server installation. The use of the workspace allows multiple clients to access the master configuration. If the same update is made by more than one client, it is possible that updates made by a scripting client will not save because there is a conflict. If this occurs, the updates will not be saved in the configuration unless you change the default save policy with the **setSaveMode** command.

The AdminConfig commands are available in both connected and local modes. If a server is currently running, it is not recommended that you run the scripting client in local mode because the configuration changes made in the local mode is not reflected in the running server configuration and vice versa. In connnected mode, the availability of the AdminConfig commands depend on the type of server to which a scripting client is connected in a Network Deployment installation.

The AdminConfig commands are available only if a scripting client is connected to a deployment manager. When connected to a node agent or an application server, the AdminConfig commands will not be available because the configuration for these server processes are copies of the master configuration that resides in the deployment manager. The copies are created in a node machine when configuration synchronization occurs between the deployment manager and the node agent. You should make configuration changes to the server processes by connecting a scripting client to a deployment manager. For this reason, to change a configuration, do not run a scripting client in local mode on a node machine. It is not a supported configuration.

The following steps provide a general method to update a configuration object:

- Identify the configuration type and the corresponding attributes.
- Query an existing configuration object to obtain a configuration ID to use.
- Modify the existing configuration object or create a one.
- Save the configuration.

To see a list of all available commands for the AdminConfig object, see the Commands for the AdminConfig object article or you can also use the **Help** command, for example:

Using Jacl:

```
$AdminConfig help
```

Using Jython:

```
print AdminConfig.help()
```

Creating configuration objects using the wsadmin tool:

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

Perform this task if you want to create an object. To create new objects from the default template, use the **create** command. Alternatively, you can create objects using an existing object as a template with the **createUsingTemplate** command.

1. Use the AdminConfig object **listTemplates** command to list available templates:
   - Using Jacl:
     ```
     $AdminConfig listTemplates JDBCProvider
     ```

- Using Jython:

  ```
  AdminConfig.listTemplates('JDBCProvider')
  ```

  where:

`$`	is a Jacl operator for substituting a variable name with its value
`AdminConfig`	is an object representing the WebSphere Application Server configuration
`listTemplates`	is an AdminConfig command
`JDBCProvider`	is an object type

2. Assign the ID string that identifies the existing object to which the new object is added. You can add the new object under any valid object type. The following example uses a node as the valid object type:
   - Using Jacl:

     ```
     set n1 [$AdminConfig getid /Node:mynode/]
     ```

   - Using Jython:

     ```
     n1 =  AdminConfig.getid('/Node:mynode/')
     ```

   where:

`set`	is a Jacl command
`$`	is a Jacl operator for substituting a variable name with its value
`n1`	is a variable name
`AdminConfig`	is an object representing the WebSphere Application Server configuration
`getid`	is an AdminConfig command
`Node`	is an object type
`mynode`	is the host name of the node where the new object is added

3. Specify the template that you want to use:
   - Using Jacl:

     ```
     set t1 [$AdminConfig listTemplates JDBCProvider "DB2 JDBC Provider (XA)"]
     ```

   - Using Jython:

     ```
     t1 = AdminConfig.listTemplates('JDBCProvider', 'DB2 JDBC Provider (XA)')
     ```

   where:

`set`	is a Jacl command
`$`	is a Jacl operator for substituting a variable name with its value
`t1`	is a variable name
`AdminConfig`	is an object representing the WebSphere Application Server configuration
`listTemplates`	is an AdminConfig command
`JDBCProvider`	is an object type
`DB2 JDBC Provider (XA)`	is the name of the template to use for the new object

   If you supply a string after the name of a type, you get back a list of templates with display names that contain the string you supplied. In this example, the AdminConfig **listTemplates** command returns the JDBCProvider template whose name matches *DB2 JDBC Provider (XA)*. This example assumes that the variable that you specify here only holds one template configuration ID. If the environment contains multiple templates with the same string, for example, *DB2 JDBC Provider (XA)*, the variable will hold the configuration IDs of all of the templates. Be sure to identify the specific template that you want to use before you perform the next step, creating an object using a template.

4. Create the object with the following command:
   - Using Jacl:
     ```
     $AdminConfig createUsingTemplate JDBCProvider $n1 {{name newdriver}} $t1
     ```
   - Using Jython:
     ```
     AdminConfig.createUsingTemplate('JDBCProvider', n1, [['name', 'newdriver']], t1)
     ```

   where:

$	is a Jacl operator for substituting a variable name with its value
AdminConfig	is an object representing the WebSphere Application Server configuration
createUsingTemplate	is an AdminConfig command
JDBCProvider	is an object type
n1	evaluates the ID of the host node specified in step number 3
name	is an attribute of JDBCProvider objects
newdriver	is the value of the name attribute
t1	evaluates the ID of the template specified in step number 4

   All **create** commands use a template unless there are no templates to use. If a default template exists, the command creates the object.

5. Save the configuration changes. See the “Saving configuration changes with the wsadmin tool” on page 232 article for more information.
6. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 article for more information.

Interpreting the output of the AdminConfig attributes command using scripting:

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

The **attributes** command is a wsadmin tool on-line help feature. When you issue the **attributes** command, the information that displays does not represent a particular configuration object. It represents information about configuration object types, or object metadata. This article discusses how to interpret the attribute type display.

- Simple attributes

  Using Jacl:
  ```
  $AdminConfig attributes ExampleType1
  "attr1 String"
  ```
  Types do not display as fully qualified names. For example, `String` is used for `java.lang.String`. There are no ambiguous type names in the model. For example, `x.y.ztype` and `a.b.ztype`. Using only the final portion of the name is possible, and it makes the output easier to read.
- Multiple attributes

  Using Jacl:

```
$AdminConfig attributes ExampleType2
"attr1 String" "attr2 Boolean" "attr3 Integer"
```

All input and output for the scripting client takes place with strings, but `attr2 Boolean` indicates that `true` or `false` are appropriate values. The `attr3 Integer` indicates that string representations of integers (″42″) are needed. Some attributes have string values that can take only one of a small number of predefined values. The wsadmin tool distinguishes these values in the output by the special type name ENUM, for example:

Using Jacl:

```
$AdminConfig attributes ExampleType3
"attr4 ENUM(ALL, SOME, NONE)"
```

where: `attr4` is an ENUM type. When you query or set the attribute, one of the values is `ALL`, `SOME`, or `NONE`. The value `A_FEW` results in an error.

- Nested attributes

  Using Jacl:

  ```
  $AdminConfig attributes ExampleType4
  "attr5 String" "ex5 ExampleType5"
  ```

  The `ExampleType4` object has two attributes: a string, and an `ExampleType5` object. If you do not know what is contained in the ExampleType5 object, you can use another **attributes** command to find out. The **attributes** command displays only the attributes that the type contains directly. It does not recursively display the attributes of nested types.

- Attributes that represent lists

  The values of these attributes are object lists of different types. The `*` character distinguishes these attributes, for example:

  Using Jacl:

  ```
  $AdminConfig attributes ExampleType5
  "ex6 ExampleType6*"
  ```

  In this example, objects of the `ExampleType5` type contain a single attribute, `ex6`. The value of this attribute is a list of `ExampleType6` type objects.

- Reference attributes

  An attribute value that references another object. You cannot change these references using modify commands, but these references display because they are part of the complete representation of the type. Distinguish reference attributes using the `@` sign, for example:

  Using Jacl:

  ```
  $AdminConfig attributes ExampleType6
  "attr7 Boolean" "ex7 ExampleType7@"
  ```

  `ExampleType6` objects contain references to `ExampleType7` type objects.

- Generic attributes

  These attributes have generic types. The values of these attributes are not necessarily this generic type. These attributes can take values of several different specific types. When you use the AdminConfig attributes command to display the attributes of this object, the various possibilities for specific types are shown in parentheses, for example:

  Using Jacl:

  ```
  $AdminConfig attributes ExampleType8
  "name String" "beast AnimalType(HorseType, FishType, ButterflyType)"
  ```

  In this example, the `beast` attribute represents an object of the generic `AnimalType`. This generic type is associated with three specific subtypes. The wsadmin tool gives these subtypes in parentheses after the name of the base type. In any particular instance of `ExampleType8`, the `beast` attribute can have a value of `HorseType`, `FishType`, or `ButterflyType`. When you specify an attribute in this way, using a modify or create command, specify the type of `AnimalType`. If you do not specify the `AnimalType`, a generic `AnimalType` object is assumed (specifying the generic type is possible and legitimate). This is done by specifying `beast:HorseType` instead of `beast`.

Specifying configuration objects using the wsadmin tool:

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

To manage an existing configuration object, identify the configuration object and obtain configuration ID of the object to be used for subsequent manipulation.

1. Obtain the configuration ID with one of the following ways:
   - Obtain the ID of the configuration object with the **getid** command, for example:
     - Using Jacl:

       ```
       set var [$AdminConfig getid /type:name/]
       ```
     - Using Jython:

       ```
       var = AdminConfig.getid('/type:name/')
       ```

     where:

`set`	is a Jacl command
`var`	is a variable name
`$`	is a Jacl operator for substituting a variable name with its value
`AdminConfig`	is an object representing the WebSphere Application Server configuration
`getid`	is an AdminConfig command
`/type:`*`name`*`/`	is the hierarchical containment path of the configuration object
`type`	is the object type **Note:** The name of the object type that you input here is the one based on the XML configuration files and does not have to be the same name that the administrative console displays.
`name`	is the optional name of the object

     You can specify multiple `/type:name/` in the string, for example, `/type:name/type:name/type:name/`. If you just specify the type in the containment path without the name, include the colon, for example, `/type:/`. The containment path must be a path containing the correct hierarchical order. For example, if you specify `/Server:server1/Node:node/` as the containment path, you will not receive a valid configuration ID because `Node` is parent of `Server` and should come before `Server` in the hierarchy.

     This command returns all the configuration IDs that match the representation of the containment and assigns them to a variable.

     To look for all the server configuration IDs resided in mynode, use the following example:
     - Using Jacl:

       ```
       set nodeServers [$AdminConfig getid /Node:mynode/Server:/]
       ```
     - Using Jython:

       ```
       nodeServers = AdminConfig.getid('/Node:mynode/Server:/')
       ```

     To look for server1 configuration ID resided in mynode, use the following example:
     - Using Jacl:

       ```
       set server1 [$AdminConfig getid /Node:mynode/Server:server1/]
       ```
     - Using Jython:

       ```
       server1 = AdminConfig.getid('/Node:mynode/Server:server1/')
       ```

     To look for all the server configuration IDs, use the following example:
     - Using Jacl:

```
set servers [$AdminConfig getid /Server:/]
```

– Using Jython:

```
servers = AdminConfig.getid('/Server:/')
```

- Obtain the ID of the configuration object with the **list** command, for example:
  - Using Jacl:

    ```
    set var [$AdminConfig list type]
    ```

    or

    ```
    set var [$AdminConfig list type scopeId]
    ```

  - Using Jython:

    ```
    var = AdminConfig.list('type')
    ```

    or

    ```
    var = AdminConfig.list('type', 'scopeId')
    ```

  where:

`set`	is a Jacl command
`var`	is a variable name
`$`	is a Jacl operator for substituting a variable name with its value
`AdminConfig`	is an object representing the WebSphere Application Server configuration
`list`	is an AdminConfig command
`type`	is the object type **Note:** The name of the object type that you input here is the one based on the XML configuration files and does not have to be the same name that the administrative console displays.
`scopeId`	is the configuration ID of a cell, node, or server object

  This command returns a list of configuration object IDs of a given type. If you specify the *scopeId*, the list of objects returned is within the scope specified. The list returned is assigned to a variable.

  To look for all the server configuration IDs, use the following example:
  - Using Jacl:

    ```
    set servers [$AdminConfig list Server]
    ```

  - Using Jython:

    ```
    servers = AdminConfig.list('Server')
    ```

  To look for all the server configuration IDs in *mynode*, use the following example:
  - Using Jacl:

    ```
    set scopeid [$AdminConfig getid /Node:mynode/]
    set nodeServers [$AdminConfig list Server $scopeid]
    ```

  - Using Jython:

    ```
    scopeid = AdminConfig.getid('/Node:mynode/')
    nodeServers = AdminConfig.list('Server', scopeid)
    ```

2. If there are more than more configuration IDs returned from the **getid** or **list** command, the IDs are returned in a list syntax. One way to retrieve a single element from the list is to use the **lindex** command. The following example retrieves the first configuration ID from the server object list:
   - Using Jacl:

     ```
     set allServers [$AdminConfig getid /Server:/]
     set aServer [lindex $allServers 0]
     ```

   - Using Jython:

```
allServers = AdminConfig.getid('/Server:/')

 # get line separator
 import  java
 lineSeparator = java.lang.System.getProperty('line.separator')

 arrayAllServers = allServers.split(lineSeparator)
 aServer = arrayAllServers[0]
```

For other ways to manipulate the list and then perform pattern matching to look for a specified configuration object, refer to the Jacl syntax.

You can now use the configuration ID in any subsequent AdminConfig commands that require a configuration ID as a parameter.

Listing attributes of configuration objects using the wsadmin tool:

Before starting this task, the wsadmin tool must be running. See the "Starting the wsadmin scripting client" on page 250 article for more information.

Perform the following steps to create a list of attributes of configuration objects:

1. List the attributes of a given configuration object type, using the **attributes** command, for example:
   - Using Jacl:
     ```
     $AdminConfig attributes type
     ```
   - Using Jython:
     ```
     AdminConfig.attributes('type')
     ```

   where:

`$`	is a Jacl operator for substituting a variable name with its value
`AdminConfig`	is an object representing the WebSphere Application Server configuration
`attributes`	is an AdminConfig command
`type`	is an object type

   This command returns a list of attributes and its data type.

   To get a list of attributes for the JDBCProvider type, use the following example command:
   - Using Jacl:
     ```
     $AdminConfig attributes JDBCProvider
     ```
   - Using Jython:
     ```
     AdminConfig.attributes('JDBCProvider')
     ```
2. List the required attributes of a given configuration object type, using the **required** command, for example:
   - Using Jacl:
     ```
     $AdminConfig required type
     ```
   - Using Jython:
     ```
     AdminConfig.required('type')
     ```

   where:

`$`	is a Jacl operator for substituting a variable name with its value
`AdminConfig`	is an object representing the WebSphere Application Server configuration

required	is an AdminConfig command
type	is an object type

This command returns a list of required attributes.

To get a list of required attributes for the JDBCProvider type, use the following example command:

- Using Jacl:

```
$AdminConfig required JDBCProvider
```

- Using Jython:

```
AdminConfig.required('JDBCProvider')
```

3. List attributes with defaults of a given configuration object type, using the **defaults** command, for example:

- Using Jacl:

```
$AdminConfig defaults type
```

- Using Jython:

```
AdminConfig.defaults('type')
```

where:

$	is a Jacl operator for substituting a variable name with its value
AdminConfig	is an object representing the WebSphere Application Server configuration
defaults	is an AdminConfig command
type	is an object type

This command returns a list of all attributes, types, and defaults.

To get a list of attributes with defaults displayed for the JDBCProvider type, use the following example command:

- Using Jacl:

```
$AdminConfig defaults JDBCProvider
```

- Using Jython:

```
AdminConfig.defaults('JDBCProvider')
```

Modifying configuration objects with the wsadmin tool:

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

Perform the following steps to modify a configuration object:

1. Retrieve the configuration ID of the objects that you want to modify, for example:

- Using Jacl:

```
set jdbcProvider1 [$AdminConfig getid /JDBCProvider:myJdbcProvider/]
```

- Using Jython:

```
jdbcProvider1 = AdminConfig.getid('/JDBCProvider:myJdbcProvider/')
```

where:

set	is a Jacl command
jdbcProvider1	is a variable name
$	is a Jacl operator for substituting a variable name with its value

AdminConfig	is an object representing the WebSphere Application Server configuration
getid	is an AdminConfig command
/JDBCProvider:*myJdbcProvider*/	is the hierarchical containment path of the configuration object
JDBCProvider	is the object type
myJdbcProvider	is the optional name of the object

2. Show the current attribute values of the configuration object with the show command, for example:
   - Using Jacl:
     ```
     $AdminConfig show $jdbcProvider1
     ```
   - Using Jython:
     ```
     AdminConfig.show(jdbcProvider1)
     ```

   where:

$	is a Jacl operator for substituting a variable name with its value
AdminConfig	is an object representing the WebSphere Application Server configuration
show	is an AdminConfig command
jdbcProvider1	evaluates to the ID of host node specified in step number 2

3. Modify the attributes of the configuration object, for example:
   - Using Jacl:
     ```
     $AdminConfig modify $jdbcProvider1 {{description "This is my new description"}}
     ```
   - Using Jython list:
     ```
     AdminConfig.modify(jdbcProvider1, [['description', "This is my new description"]])
     ```
   - Using Jython string:
     ```
     AdminConfig.modify(jdbcProvider1, '[[description "This is my new description"]]')
     ```

   where:

$	is a Jacl operator for substituting a variable name with its value
AdminConfig	is an object representing the WebSphere Application Server configuration
modify	is an AdminConfig command
jdbcProvider1	evaluates to the ID of host node specified in step number 3
description	is an attribute of server objects
This is my new description	is the value of the description attribute

   You can also modify several attributes at the same time. For example:
   - Using Jacl:
     ```
     {{name1 val1} {name2 val2} {name3 val3}}
     ```
   - Using Jython list:
     ```
     [['name1', 'val1'], ['name2', 'val2'], ['name3', 'val3']]
     ```
   - Using Jython string:

```
'[[name1 val1] [name2 val2] [name3 val3]]'
```

4. Save the configuration changes. See the “Saving configuration changes with the wsadmin tool” on page 232 article for more information.
5. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 article for more information.

Removing configuration objects with the wsadmin tool:

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

Use this task to delete a configuration object from the configuration repository. This action only affects the configuration. If there is a running instance of a configuration object when you remove the configuration, the change has no effect on the running instance.

1. Assign the ID string that identifies the server you want to remove:

   Using Jacl:

   ```
   set s1 [$AdminConfig getid /Node:mynode/Server:myserver/]
   ```

   Using Jython:

   ```
   s1 = AdminConfig.getid('/Node:mynode/Server:myserver/')
   ```

   where:

set	is a Jacl command
s1	is a variable name
$	is a Jacl operator for substituting a variable name with its value
AdminConfig	is an object representing the WebSphere Application Server configuration
getid	is an AdminConfig command
Node	is an object type
mynode	is the host name of the node from which the server is removed
Server	is an object type
myserver	is the name of the server to remove

2. Remove the configuration object. For example:
   - Using Jacl:

     ```
     $AdminConfig remove $s1
     ```

   - Using Jython:

     ```
     AdminConfig.remove(s1)
     ```

   where:

$	is a Jacl operator for substituting a variable name with its value
AdminConfig	is an object representing the WebSphere Application Server configuration
remove	is an AdminConfig command
s1	evaluates the ID of the server specified in step number 2

3. Save the configuration changes. See the “Saving configuration changes with the wsadmin tool” on page 232 article for more information.
4. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 article for more information.

The WebSphere Application Server configuration no longer contains a specific server object. Running servers are not affected.

Changing the WebSphere Application Server configuration using wsadmin:

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information. For this task, the wsadmin scripting client must be connected to the deployment manager server in a network deployment environment.

You can use the wsadmin AdminConfig and AdminApp objects to make changes to the WebSphere Application Server configuration. The purpose of this article is to illustrate the relationship between the commands used to change the configuration and the files used to hold configuration data. This discussion assumes that you have a network deployment installation, but the concepts are very similar for a WebSphere Application Server installation.

1. Set a variable for creating a server:
   - Using Jacl:
     ```
     set n1 [$AdminConfig getid /Node:mynode/]
     ```
   - Using Jython:
     ```
     n1 = AdminConfig.getid('/Node:mynode/')
     ```

   where:

set	is a Jacl command
n1	is a variable name
$	is a Jacl operator for substituting a variable name with its value
AdminConfig	is an object representing the WebSphere Application Server configuration
getid	is an AdminConfig command
Node	is the object type
mynode	is the name of the object that will be modified

2. Create a server with the following command:
   - Using Jacl:
     ```
     set serv1 [$AdminConfig create Server $n1 {{name myserv}}]
     ```
   - Using Jython list:
     ```
     serv1 = AdminConfig.create('Server', n1, [['name', 'myserv']])
     ```
   - Using Jython string:
     ```
     serv1 = AdminConfig.create('Server', n1, '[[name myserv]]')
     ```

   where:

set	is a Jacl command
serv1	is a variable name
$	is a Jacl operator for substituting a variable name with its value

`AdminConfig`	is an object representing the WebSphere Application Server configuration
`create`	is an AdminConfig command
`Server`	is an AdminConfig object
`n1`	evaluates to the ID of host node specified in step number 2
`name`	is an attribute
`myserv`	is the value of the name attribute

After this command completes, some new files can be seen in a workspace used by the deployment manager server on behalf of this scripting client. A workspace is a temporary repository of configuration information that administrative clients use. Any changes made to the configuration by an administrative client are first made to this temporary workspace. For scripting, only when a **save** command is invoked on the AdminConfig object, these changes are transferred to the real configuration repository. Workspaces are kept in the `wstemp` subdirectory of a WebSphere Application Server installation.

3. Make a configuration change to the server with the following command:
   - Using Jacl:
     ```
     $AdminConfig modify $serv1 {{stateManagement {{initialState STOP}}}}
     ```
   - Using Jython list:
     ```
     AdminConfig.modify(serv1, [['stateManagement', [['initialState', 'STOP']]]])
     ```
   - Using Jython string:
     ```
     AdminConfig.modify(serv1, '[[stateManagement  [[initialState  STOP]]]]')
     ```

   where:

`$`	is a Jacl operator for substituting a variable name with its value
`AdminConfig`	is an object representing the WebSphere Application Server configuration
`modify`	is an AdminConfig command
`serv1`	evaluates to the ID of host node specified in step number 3
`stateManagement`	is an attribute
`initialState`	is a nested attribute within the stateManagement attribute
`STOP`	is the value of the initialState attribute

This command changes the initial state of the new server. After this command completes, one of the files in the workspace is changed.

4. Install an application on the server.
5. Save the configuration changes. See the “Saving configuration changes with the wsadmin tool” on page 232 article for more information.
6. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 article for more information.

Modifying nested attributes with the wsadmin tool:

The attributes for a WebSphere Application Server configuration object are often deeply nested. For example, a JDBCProvider object has an attribute factory, which is a list of the J2EEResourceFactory type objects. These objects can be DataSource objects that contain a connectionPool attribute with a ConnectionPool type that contains a variety of primitive attributes.

1. Invoke the AdminConfig object commands interactively, in a script, or use the **wsadmin -c** commands from an operating system command prompt.

2. Obtain the configuration ID of the object, for example:

   Using Jacl:

   ```
   set t1 [$AdminConfig getid /DataSource:TechSamp/]
   ```

   Using Jython:

   ```
   t1=AdminConfig.getid('/DataSource:TechSamp/')
   ```

   where:

set	is a Jacl command
t1	is a variable name
$	is a Jacl operator for substituting a variable name with its value
AdminConfig	is an object representing the WebSphere Application Server configuration
getid	is an AdminConfig command
DataSource	is the object type
TechSamp	is the name of the object that will be modified

3. Modify one of the object parents and specify the location of the nested attribute within the parent, for example:

   Using Jacl:

   ```
   $AdminConfig modify $t1 {{connectionPool {{reapTime 2003}}}}
   ```

   Using Jython list:

   ```
   AdminConfig.modify(t1, [["connectionPool", [["reapTime", 2003]]]])
   ```

   Using Jython string:

   ```
   AdminConfig.modify(t1, '[[connectionPool [[reapTime 2003]]]]')
   ```

   where:

$	is a Jacl operator for substituting a variable name with its value
AdminConfig	is an object representing the WebSphere Application Server configuration
modify	is an AdminConfig command
t1	evaluates to the configuration ID of the datasource in step number 2
connectionPool	is an attribute
reapTime	is a nested attribute within the connectionPool attribute
2003	is the value of the reapTime attribute

4. Save the configuration by issuing an AdminConfig **save** command. For example:

   Using Jacl:

   ```
   $AdminConfig save
   ```

   Using Jython:

   ```
   AdminConfig.save()
   ```

   Use the **reset** command of the AdminConfig object to undo changes that you made to your workspace since your last save.

An alternative way to modify nested attributes is to modify the nested attribute directly, for example:

Using Jacl:

```
set techsamp [$AdminConfig getid /DataSource:TechSamp/]
set pool [$AdminConfig showAttribute $techsamp connectionPool]
$AdminConfig modify $pool {{reapTime 2003}}
```

Using Jython list:

```
techsamp=AdminConfig.getid('/DataSource:TechSamp/')
pool=AdminConfig.showAttribute(techsamp,'connectionPool')
AdminConfig.modify(pool,[['reapTime',2003]])
```

Using Jython string:

```
techsamp=AdminConfig.getid('/DataSource:TechSamp/')
pool=AdminConfig.showAttribute(techsamp,'connectionPool')
AdminConfig.modify(pool,'[[reapTime 2003]]')
```

In this example, the first command gets the configuration id of the DataSource, and the second command gets the connectionPool attribute. The third command sets the reapTime attribute on the ConnectionPool object directly.

Saving configuration changes with the wsadmin tool:

The wsadmin tool uses the workspace to hold configuration changes. You must save your changes to transfer the updates to the master configuration repository. If a scripting process ends and you have not saved your changes, the changes are discarded. Use the following commands to save the configuration changes:

- Using Jacl:

  ```
  $AdminConfig save
  ```

- Using Jython:

  ```
  AdminConfig.save()
  ```

where:

$	is a Jacl operator for substituting a variable name with its value
AdminConfig	is an object representing the WebSphere Application Server configuration
save	is an AdminConfig command

If you are using interactive mode with the wsadmin tool, you will be prompted to save your changes before they are discarded. If you are using the -c option with the wsadmin tool, changes are automatically saved.

You can use the **reset** command of the AdminConfig object to undo changes that you made to your configuration since your last save.

AdminTask object for scripted administration: Use the AdminTask object to access a set of administrative commands that provide an alternative way to access the configuration commands and the running object management commands. The administrative commands run simple and complex commands. They provide more user friendly and task-oriented commands. The administrative commands are discovered dynamically when you start a scripting client. The set of available administrative commands depends on the edition of WebSphere Application Server you install. You can use the AdminTask object commands to access these commands.

Administrative commands are grouped based on their function. You can use administrative command groups to find related commands. For example, the administrative commands that are related to server management are grouped into a server management command group. The administrative commands that are related to the security management are grouped into a security management command group. An administrative command can be associated with multiple command groups because it can be useful for multiple areas of system management. Both administrative commands and administrative command groups are uniquely identified by their name.

Two run modes are always available for each administrative command, namely the *batch* and *interactive mode*. When you use an administrative command in interactive mode, you go through a series of steps to collect your input interactively. This process provides users a text-based wizard and a similar user experience to the wizard in the administrative console. You can also use the help command to obtain help for any administrative command and the AdminTask object.

The administrative commands do not replace any existing configuration commands or running object management commands but provide a way to access these commands and organize the inputs. Depending on the administrative command, it can be available in connected or local mode. The set of available administrative commands is determined when you start a scripting client in connected or local mode. If a server is running, it is not recommended that you run the scripting client in local mode because any configuration changes made in local mode are not reflected in the running server configuration and vice versa. If you save a conflicting configuration, you could corrupt the configuration. In a deployment manager environment, configuration updates are available only if a scripting client is connected to a deployment manager. When connected to a node agent or a managed application server, you will not be able to update the configuration because the configuration for these server processes are copies of the master configuration which resides in the deployment manager. The copies are created on a node machine when a configuration synchronization occurs between the deployment manager and the node agent. Make configuration changes to the server processes by connecting a scripting client to a deployment manager. For this reason, to change a configuration, do not run a scripting client in local mode on a node machine. It is not a supported configuration.

Obtaining online help using scripting:

There are three levels of online help available with the administrative commands. The top level help provides general information for the AdminTask object and the commands associated with it. The second level help provides information about all of the available administrative commands and command groups. The third level help provides specific help on a command group, a command, or a step. Command group specific help provides descriptions for the command group that you specify and the commands that belong to the associated group. Command specific help provides description for the specified command, its parameters, and steps, if any. Step specific help provides a description for the specified step and the associated parameters. For command and step specific help, required parameters are marked with a `*` in the help output.

- To obtain general help, use the following examples:

  Using Jacl:

  ```
  $AdminTask help
  ```

  Using Jython:

  ```
  print AdminTask.help()
  ```

  Example output:

  ```
  WASX8001I: The AdminTask object enables the execution of available
             admin commands. AdminTask commands operate in two modes:
             the default mode is one which AdminTask communicates with the
             WebSphere server to accomplish its task. A local mode is also
             available in which no server communication takes place. The local
             mode of operation is invoked by bringing up the scripting client
             using the command line "-conntype NONE" option or setting the
             "com.ibm.ws.scripting.connectiontype=NONE" property in
             wsadmin.properties file.

  The number of admin commands varies and depends on your WebSphere install.
  Use the following help commands to obtain a list of supported commands
  and their parameters:
  ```

```
help -commands
      list all the admin commands
help -commandGroups
      list all the admin command groups
help commandName
      display detailed information for
        the specified command
help commandName stepName
      display detailed information for
      the specified step belonging to
      the specified command
help commandGroupName
      display detailed information for
               the specified command group

There are various flavors to invoke an admin command:

commandName
      invokes an admin command that does not require any argument.

commandName targetObject
      invokes an admin command with the specified target object
      string, for example, the configuration object name of a
      resource adapter. The expected target object varies with
      the admin command invoked. Use help command to get
      information on the target object of an admin command.

commandName options
      invokes an admin command with the specified option
      strings. This invocation syntax is used to invoke an
               admin command that does not require a target object. It
               is also used to enter interactive mode if "-interactive"
               mode is included in the options string.

commandName targetObject options
      invokes an admin command with the specified target
      object and options strings. If "-interactive" is
      included in the options string, then interactive mode
      is entered. The target object and options strings vary
      depending on the admin command invoked. Use help
      command to get information on the target
      object and options.
```

- To list the available command groups, use the following examples:

  Using Jacl:

  ```
  $AdminTask help -commandGroups
  ```

  Using Jython:

  ```
  print AdminTask.help('-commandGroups')
  ```

  Example output:

  ```
  WASX8005I: Available admin command groups:

  ClusterConfigCommands - Commands for configuring application
  server clusters and cluster members.
  JCAManagement - A group of admin commands that helps to configure
  Java2 Connector Architecture(J2C) related resources.
  ```

- To list the available commands, use the following examples:

  Using Jacl:

  ```
  $AdminTask help -commands
  ```

  Using Jython:

  ```
  print AdminTask.help('-commands')
  ```

  Example output:

```
WASX8004I: Available administrative commands:

copyResourceAdapter - copy the specified J2C resource adapter to the specified scope
createCluster - Creates a new application server cluster.
createClusterMember - Creates a new member of an application server cluster.
createJ2CConnectionFactory - Create a J2C connection factory
deleteCluster - Delete the configuration of an application server cluster.
deleteClusterMember - Deletes a member from an application server cluster.
listConnectionFactoryInterfaces - list all of the
defined connection factory interfaces on the
specified J2C resource adapter.
listJ2CConnectionFactories - List J2C connection factories that have a specified
connection factory interface defined in the specified J2C resouce adapter
createJ2CAdminObject - Create a J2C administrative object.
listAdminObjectInterfaces - List all the defined administrative object interfaces
on the specified J2C resource adapter.
interface on the specified J2C resource adapter.
listJ2CAdminObjects - List the J2C administrative objects that have a specified
administrative object interface defined in the specified J2C resource adapter.
createJ2CActivationSpec - Create a J2C activation specification.
listMessageListenerTypes - list all of the defined messageListener
type on the specified J2C resource adapter.
listJ2CActivationSpecs - List the J2C activation specifications that have a
specified message listener type defined in the specified J2C resource adapter.
```

- To obtain help about a command group, use the following examples:

  Using Jacl:

  ```
  $AdminTask help JCAManagement
  ```

  Using Jython:

  ```
  print AdminTask.help('JCAManagement')
  ```

  Example output:

  ```
  WASX8007I: Detailed help for command group: JCAManagement

  Description: A group of administrative commands that help to
  configure Java 2 Connector Architecture (J2C)-related resources.

  Commands:
  createJ2CConnectionFactory - Create a J2C connection factory
  listConnectionFactoryInterfaces - list all of the defined connection
  factory interfaces on the specified J2C resource adapter.
  listJ2CConnectionFactories - List J2C connection factories that have
  a specified connection factory interface defined in the
  specified J2C resouce adapter.
  createJ2CAdminObject - Create a J2C administrative object.
  listAdminObjectInterfaces - List all the defined administrative
  object interfaces on the specified J2C resource adapter.
  listJ2CAdminObjects - List the J2C administrative objects that have a
  specified adminstrative object interface defined in the
  specified J2C resource adapter.
  createJ2CActivationSpec - Create a J2C activation specification.
  listMessageListenerTypes - list all of the defined
  message listener types on the specified J2C resource adapter.
  listJ2CActivationSpecs - List the J2C activation specifications that
  have a specified message listener type defined in the
  specified J2C resource adapter.
  copyResourceAdapter - copy the specified J2C resource
  adapter to the specified scope.
  ```

- Obtaining help about an administrative command:

  Using Jacl:

  ```
  $AdminTask help createJ2CConnectionFactory
  ```

  Using Jython:

  ```
  print AdminTask.help('createJ2CConnectionFactory')
  ```

Example output:

```
WASX8006I: Detailed help for command: createJ2CConnectionFactory

Description: Create a J2C connection factory

*Target object: The parent J2C resource adapter of the created J2C connection factory.

Arguments:
*connectionFactoryInterface - A connection factory interface that is defined in the deployment
 description of the parent J2C resource adapter.
*name - The name of the J2C connection factory.
*jndiName - The JNDI name of the created J2C connection factory.
description - The description for the created J2C connection factory.
authDataAlias - the authentication data alias of the created J2C connection factory.

Steps:
None
```

In the command specific help output previously listed, an administrative command is divided into three input areas: target object, arguments, and steps. Each area can require input depending on the administrative command. If an area requires input, each input is described by its name and a description; except for the target object area which only contains the description of the target object. When you use an administrative command in batch mode, you can use any input name that resides in the argument area as the argument name. If an input is required, a * will be before the name. If an area does not require an input, it is marked None. The following example uses the help output for the **createJ2CConnectionFactory** command:

– Target object area requires the configuration object name of a J2CResourceAdapter to be provided.

– In the arguments area, there are five inputs with three being required inputs. The argument names are connectionFactoryInterface, name, jndiName, description, and authDataAlias. These names are used as the parameter names in the option string to execute an admin command in batch mode, for example:

  ```
  -connectionFactoryInterface javax.resource.cci.ConnectionFactory -name newConnectionFactory
   -jndiName CF/newConnectionFactory
  ```

  See “Administrative command invocation syntax” on page 623 for more information about specifying argument options.

– There is no step associated with this administrative command.

- Obtain help on a command step.

  Step specific help provides the following:

  – A description for the command step.

  – Information indicating if this step supports collection. A collection includes objects of the same type. In a command step, a collection contains objects that have the same set of parameters.

  – Information regarding each step parameter with its name and description. If a step parameter is required, a * exists in front of the name.

  The following example obtain help on a command step:

  Using Jacl:

  ```
  $AdminTask help createCluster clusterConfig
  ```

  Using Jython:

  ```
  print AdminTask.help('createCluster', 'clusterConfig')
  ```

  Example output:

  ```
  WASX8013I: Detailed help for step: clusterConfig

  Description: Specifies the configuration of the new server cluster.

  Collection: No
  ```

```
Arguments:
  *clusterName - Name of server cluster.
  preferLocal - Enables node-scoped routing optimization for the cluster.
```

This example indicates the following:

– It does not support collection. Only one set of parameter values for the clusterName and perferLocal parameters is allowed.

– It contains two input arguments with one argument indicated as required. The required arguments is clusterName and the non-required parameter is perferLocal. The syntax to provide step parameter values is different from the command argument values. You have to provide all argument values of a step and provide them in the exact order as displayed in the step specific help. For any optional argument that you do not want to specify a value, put double quotes ("") in place of a value. If a command step is a collection type, for example, it can contain multiple objects where each object has the same set of arguments, you can specify multiple objects with each object enclosed by its own pair of braces. To execute an administrative command in batch mode and to include this step in the option string, use the following syntax:

  Using Jacl:

  ```
  -clusterConfig {{newCluster false}}
  ```

  Using Jython:

  ```
  -clusterConfig [[newCluster false]]
  ```

  See "Administrative command invocation syntax" on page 623 for more information about specifying parameter options.

Invoking an administrative command in batch mode:

Perform the following steps to invoke an administrative command in batch mode. To invoke an administrative command in interactive mode, see "Invoking an administrative command in interactive mode" on page 242.

1. Invoke the AdminTask object commands interactively, in a script, or use the **wsadmin -c** command from an operating system command prompt.
2. Issue one of the following commands:
   - If an administrative command does not have a target object and an argument, use the following command:

     Using Jacl:

     ```
     $AdminTask commandName
     ```

     Using Jython:

     ```
     AdminTask.commandName()
     ```

     where:

$	is a Jacl operator for substituting a variable name with its value
AdminTask	is an object allowing administrative command management
commandName	is the name of the administrative command being invoked

   - If an administrative command includes a target object but does not include any arguments or steps, use the following command:

     Using Jacl:

     ```
     $AdminTask commandName targetObject
     ```

     Using Jython:

     ```
     AdminTask.commandName(targetObject)
     ```

where:

$	is a Jacl operator for substituting a variable name with its value
AdminTask	is an object allowing administrative command management
commandName	is the name of the administrative command being invoked
targetObject	is the target object string for the invoked administrative command. The expect target object varies with each administrative command. View the online help for the invoked administrative command to learn more about what you should specify as the target object.

- If an administrative command includes an argument or a step but does not include a target object, use the following command:

  Using Jacl:

  ```
  $AdminTask commandName options
  ```

  Using Jython:

  ```
  AdminTask.commandName(options)
  ```

  where:

$	is a Jacl operator for substituting a variable name with its value
AdminTask	is an object allowing administrative command management
commandName	is the name of the administrative command being invoked

options	is the option string for the invoked administrative command. Depending on which administrative command you are invoking, the administrative command can have required or optional option values. The options string is different for each administrative command. View the online help for the invoked administrative command to obtain more information about which options are available. Arguments and steps listed on the online administrative command help are specified as options in the option string. Each option consists of a dash followed immediately by an option name, and then followed by an option value if the option requires a value. If the invoked administrative command includes target objects, arguments, or steps, then the –interactive option is available to enter interactive mode. For example, using the output of the following online help for listDataSource:
	`WASX8006I: Detailed help for command: exportServer`
	
	`Description: export the configuration`
	`of a server to a config archive.`
	
	`Target object: None`
	
	`Arguments:`
	`*serverName - the name of a server`
	`*nodeName - the name of a node. This parameter`
	`becomes optional if the specified server name`
	`is unique across the cell.`
	`*archive - the fully qualified file path of a config`
	`archive.`
	
	`Steps:`
	`None`
	
	Option names are specified with a dash before the names. There are three required options for this administrative command. The required options are -serverName, -nodename, and -archive. In addition, the -interactive option is available. Options are specified in the option string which is enclosed by a pair of {} in Jacl and a pair of [] in Jython.

- If an administrative command includes a target object, and arguments or steps:

  Using Jacl:

  ```
  $AdminTask commandName targetObject options
  ```

  Using Jython:

  ```
  AdminTask.commandName(targetObject, options)
  ```

  where:

`$`	is a Jacl operator for substituting a variable name with its value
`AdminTask`	is an object allowing administrative command management
commandName	is the name of the administrative command being invoked

targetObject

is the target object string for the invoked administrative command. The expected target object varies with each administrative command. View the online help for the invoked administrative command to obtain information about what to specify as a target object. For example, using the output of the following online help for createJ2CConnectionFactory:

```
WASX8006I: Detailed help for command:
createJ2CConnectionFactory

Description: Create a J2C connection factory

*Target object: The parent J2C resource adapter of
the created J2C connection factory.

Arguments:
*connectionFactoryInterface - A connection factory
interface that is defined in the deployment description
of the parent J2C resource adapter.
*name - The name of the J2C connection factory.
*jndiName - The JNDI name of the created J2C
connection factory.
description - The description for the created J2C
connection factory.
authDataAlias - the authentication data alias of the
created J2C connection factory.

Steps:
None
```

The target object is a configuration object name of a J2CResourceAdapter.

options

is the option string for the invoked administrative command. Depending on which administrative command you are invoking, the administrative command can have required or optional option values. The options string is different for each administrative command. View the online help for the invoked administrative command to obtain more information about which options are available. Arguments and steps listed on the online administrative command help are specified as options in the option string. Each option consists of a dash followed immediately by an option name, and then followed by an option value if the option requires a value. If the invoked administrative command includes target objects, arguments, or steps, then the –interactive option is available to enter interactive mode. For example, using the output of the following online help for listDataSource:

```
WASX8006I: Detailed help for command:
createJ2CConnectionFactory

Description: Create a J2C connection factory

*Target object: The parent J2C resource adapter
of the created J2C connection factory.

Arguments:
*connectionFactoryInterface - A connection factory
interface that is defined in the deployment description
of the parent J2C resource adapter.
*name - The name of the J2C connection factory.
*jndiName - The JNDI name of the created J2C
connection factory.
description - The description for the created J2C
connection factory.
authDataAlias - the authentication data alias of the
created J2C connection factory.

Steps:
None
```

Option names are specified with a dash before the names. There are three required options for this administrative command. The required options are -connectionFactoryInterface, -name, and -jndiName. There are two optional options. They are -description and -authDataAlias. In addition, the -interactive option is available. Options are specified in the option string which is enclosed by a pair of {} in Jacl and a pair of [] in Jython.

- The following example invokes an administrative command with no target object, argument, or step:

  Using Jacl:

  ```
  $AdminTask listNodes
  ```

  Using Jython:

  ```
  print AdminTask.listNodes()
  ```

  Example output:

  ```
  myNode
  ```

- The following example invokes an administrative command with a target object string:

  Using Jacl:

```
set s1 [$AdminConfig getid /Server:server1/]
$AdminTask showServerInfo $s1
```

Using Jython:

```
s1 = AdminConfig.getid('/Server:server1/')
print AdminTask.showServerInfo(s1)
```

Example output:

```
{cell myCell}
{serverType APPLICATION_SERVER}
{com.ibm.websphere.baseProductVersion 6.0.0.0}
{node myNode}
{server server1}
```

- The following example invokes an administrative command with an option string:

  Using Jacl:

  ```
  $AdminTask getNodeMajorVersion {-nodeName myNode}
  ```

  Using Jython:

  ```
  print AdminTask.getNodeMajorVersion('[-nodeName myNode]')
  ```

  Example output:

  ```
  6
  ```

- The following example invokes an administrative command with target object and non-step option strings:

  Using Jacl:

  ```
  set ra [$AdminConfig getid /J2CResourceAdapter:myResourceAdapter/]
  $AdminTask createJ2CConnectionFactory
  $ra {-name myJ2CCF -jndiName j2c/cf -connectionFactoryInterface javax.resource.cci.ConnectionFactory}
  ```

  Using Jython:

  ```
  ra = AdminConfig.getid('/J2CResourceAdapter:myResourceAdapter/')
  AdminTask.createJ2CConnectionFactory(ra,
  '[-name myJ2CCF -jndiName j2c/cf -connectionFactoryInterface javax.resource.cci.ConnectionFactory]')
  ```

  Example output:

  ```
  myJ2CCF(cells/myCell/nodes/myNode|resources.xml#J2CConnectionFactory_1069690568269)
  ```

- The following example invokes an administrative command with a target object and a step option:

  Using Jacl:

  ```
  set serverCluster [$AdminConfig getid /ServerCluster:myCluster/]
  $AdminTask createClusterMember $serverCluster {-memberConfig {{myNode myClusterMember "" "" false false}}}
  ```

  Using Jython:

  ```
  serverCluster = AdminConfig.getid('/ServerCluster:myCluster/')
  AdminTask.createClusterMember(serverCluster, '[-memberConfig [[myNode myClusterMember "" "" false false]]]')
  ```

  Example output:

  ```
  myClusterMember(cells/myCell/nodes/myNode|cluster.xml#ClusterMember_3673839301876)
  ```

Invoking an administrative command in interactive mode:

Perform the following steps to invoke an administrative command in interactive mode. To invoke an administrative command in batch mode, see "Invoking an administrative command in batch mode" on page 237.

1. Invoke the AdminTask object commands interactively, in a script, or use the **wsadmin -c** command from an operating system command prompt.
2. Invoke an administrative command in interactive mode by issuing one of the following commands:
   - Use the following command invocation to enter interactive mode without providing another input in the command invocation:

     Using Jacl:

     ```
     $AdminTask commandName {-interactive}
     ```

Using Jython:

```
AdminTask.commandName('[-interactive]')
```

where:

$	is a Jacl operator for substituting a variable name with its value
AdminTask	is an object allowing administrative command management
commandName	is the name of the administrative command being invoked
-interactive	is the interactive option

- Use the following command invocation to enter interactive mode using an administrative command that takes a target object. You do not have to provide a target object to enter interactive mode. Target objects provided in the command invocation will be applied to the command and displayed as the current target object value during interactive prompting.

  Using Jacl:

  ```
  $AdminTask commandName targetObject {-interactive}
  ```

  Using Jython:

  ```
  AdminTask.commandName(targetObject, '[-interactive]')
  ```

  where:

$	is a Jacl operator for substituting a variable name with its value
AdminTask	is an object allowing administrative command management
commandName	is the name of the administrative command being invoked
targetObject	is the target object string for the invoked administrative command. The target object is different for each administrative command. View the online help for the invoked administrative command to learn more about what to specify as a target object.
-interactive	is the interactive option

- Use the following command invocation to enter interactive mode for an administrative command that takes options. You do not have to provide other options to enter interactive mode. Options provided in the command invocation are applied to the command and the option values will be displayed as the current values during interactive prompting.

  Using Jacl:

  ```
  $AdminTask commandName {-interactive commandOptions}
  ```

  Using Jython:

  ```
  AdminTask.commandName('[-interactive commandOptions]')
  ```

  where:

$	is a Jacl operator for substituting a variable name with its value
AdminTask	is an object allowing administrative command management
commandName	is the name of the administrative command being invoked
-interactive	is the interactive option

commandOptions	is the command option available for the associated administrative command. Available command options are different for each administrative command. View the online help for the invoked administrative command to obtain more information about which options are available. Arguments and steps listed on the online administrative command help are specified as command options. Each option consists of a dash followed immediately by an option name, and then followed by an option value if the option requires a value. For example, using the output of the following online help for createJ2CConnectionFactory:
	`WASX8006I: Detailed help for command: createJ2CConnectionFactory`
	
	`Description: Create a J2C connection factory`
	
	`*Target object: The parent J2C resource adapter of the created J2C co`
	
	`Arguments:`
	`*connectionFactoryInterface - A connection factory interface that is`
	`*name - The name of the J2C connection factory.`
	`*jndiName - The JNDI name of the created J2C connection factory.`
	`description - The description for the created J2C connection factory.`
	`authDataAlias - the authentication data alias of the created J2C conn`
	
	`Steps:`
	`None`
	
	In this example, there are five options available:
	• -connectionFactoryInterface
	• -name
	• -jndiName
	• -description
	• -authDataAlias
	Each requires an option value. Only three of the options are required and are denoted with a `*`.

- Use the following command invocation to enter interactive mode for an administrative command that has a target object and options. You do not have to specify a target object to enter interactive mode. The values specified are applied to the command before the command data is displayed. As a result, the values specified will be displayed as the current values during interactive prompting.

  Using Jacl:

  ```
  $AdminTask commandName targetObject {-interactive commandOptions}
  ```

  Using Jython:

  ```
  AdminTask.commandName(targetObject, '[-interactive commandOptions]')
  ```

  where:

`$`	is a Jacl operator for substituting a variable name with its value
`AdminTask`	is an object allowing administrative command management
commandName	is the name of the administrative command being invoked

targetObject	is the target object string for the invoked admin command. The expect target object varies with each admin command. Consult the on-line help on the invoked admin command to learn more about what to specify as target object.
-interactive	is the interactive option

commandOptions	is the command option available for the associated administrative command. Available command options are different for each administrative command. View the online help for the invoked administrative command to obtain more information about which options are available. Arguments and steps listed on the online administrative command help are specified as command options. Each option consists of a dash followed immediately by an option name, and then followed by an option value if the option requires a value. For example, using the output of the following online help for createJ2CConnectionFactory:
	`WASX8006I: Detailed help for command: createJ2CConnectionFactory`
	
	`Description: Create a J2C connection factory`
	
	`*Target object: The parent J2C resource adapter of the created J2C`
	
	`Arguments:`
	`*connectionFactoryInterface - A connection factory interface that`
	`*name - The name of the J2C connection factory.`
	`*jndiName - The JNDI name of the created J2C connection factory.`
	`description - The description for the created J2C connection facto`
	`authDataAlias - the authentication data alias of the created J2C co`
	
	`Steps:`
	`None`
	
	In this example, there are five options available:
	• -connectionFactoryInterface
	• -name
	• -jndiName
	• -description
	• -authDataAlias
	
	Each requires an option value. Only three of the options are required and are denoted with a *.

- The following example invokes an administrative command in interactive mode by specifying the -interactive option:

  Using Jacl:

  ```
  $AdminTask createJ2CConnectionFactory {-interactive}
  ```

  Using Jython:

  ```
  AdminTask.createJ2CConnectionFactory('[-interactive]')
  ```

  Example output:

  ```
  Create a J2C connection factory

  *The J2C resource adapter: "WebSphere Relational ResourceAdapter
  (cells/myCell/nodes/myNode|resources.xml#builtin_rra)"

  A connection factory
  interface (connectionFactoryInterface):javax.resource.cci.ConnectionFactory
  ```

```
*Name (name): myJ2CCF
*The JNDI name (jndiName): j2c/cf
Description (description):
authentication data alias (authDataAlias):

create J2C connection factory

F (Finish)
C (Cancel)

Select [F, C]: [F]

myJ2CCF(cells/myCell/nodes/myNode|resources.xml#J2CConnectionFactory_1069690568269)
```

- The following example invokes an administrative command using the –interactive option with a target object specified in the command invocation:

  Using Jacl:

  ```
  set ra [$AdminConfig getid /J2CResourceAdapter:myResourceAdapter/]
  $AdminTask createJ2CConnectionFactory $ra {-interactive}
  ```

  Using Jython:

  ```
  ra = AdminConfig.getid('/J2CResourceAdapter:myResourceAdapter/')
  AdminTask.createJ2CConnectionFactory(ra, '[-interactive]')
  ```

  Example output:

  ```
  Create a J2C connection factory

  *The J2C resource adapter: ["WebSphere Relational ResourceAdapter
  (cells/myCell/nodes/myNode|resources.xml#builtin_rra)"]

  A connection factory interface (connectionFactoryInterface):
  javax.resource.cci.ConnectionFactory
  *Name (name): myJ2CCF
  *The JNDI name (jndiName): j2c/cf
  Description (description):
  authentication data alias (authDataAlias):

  create J2C Connection Factory

  F (Finish)
  C (Cancel)

  Select [F, C]: [F]

  myJ2CCF(cells/myCell/nodes/myNode|resources.xml#J2CConnectionFactory_1069690568269)
  ```

- The following example invokes an administrative command using the –interactive option where both the target object and the additional command options are specified in the command invocation:

  Using Jacl:

  ```
  set ra [$AdminConfig getid /J2CResourceAdapter:myResourceAdapter/]
  $AdminTask createJ2CConnectionFactory $ra {-name myNewCF -interactive}
  ```

  Using Jython:

  ```
  ra = AdminConfig.getid('/J2CResourceAdapter:myResourceAdapter/')
  AdminTask.createJ2CConnectionFactory(ra, '[-name myNewCF -interactive]')
  ```

  Example output:

  ```
  Create a J2C connection factory

  *The J2C resource adapter: ["WebSphere Relational ResourceAdapter
  (cells/myCell/nodes/myNode|resources.xml#builtin_rra)"]

  A connection factory interface (connectionFactoryInterface):javax.resource.cci.ConnectionFactory
  *Name (name): [myNewCF]
  *The JNDI name (jndiName): j2c/cf
  Description (description):
  ```

```
authentication data alias (authDataAlias):

create J2C Connection Factory

F (Finish)
C (Cancel)

Select [F, C]: [F]

myNewCF(cells/myCell/nodes/myNode|resources.xml#J2CConnectionFactory_3839439380269)
```

Administrative command interactive mode environment: An administrative command can be run in interactive mode by providing the -interactive option in the options string when invoking the command. You can still provide other options, even when using the interactive option. The options values that are specified are applied to the command before the command data is displayed. Whether or not other options are specified, the wsadmin tool steps the user through the command to collect command information.

The general interactive flow sequence is:

1. Collect user inputs for target object and parameters
2. If the command does not include a step, the command execution menu displays to run or cancel the command.
3. If the command includes a step, the menu to select the step displays. When all the required inputs are entered, the menu includes command execution.
4. When a step is selected, if the step supports collection, then the menu to select an object in the collection displays and you can exit the step. If you exit the step, repeat steps 3-5.
5. Collect user inputs for the selected step or for an object in the collection
6. Repeat steps 4 and 5 if from the collection step menu
7. Repeat steps 3-5 if from step selection menu

Depending on what input area is enabled by an administrative command, you can go through part or all of the interactive flow sequence. If an administrative command is run in interactive mode, the syntax to run the command except for the deletion of collection object in batch mode is generated and logged as a WASX7278I message in both the interactive session and in the wsadmin trace file.

Collect user inputs for target object and parameters

The following interactive prompt is used to collect inputs for the Target object and Arguments input areas that are displayed in the command-specific help:

```
Command title

Command Description

*target object title [current or default value]:
*param1 title (param1 name) [choice1, choice2, ...]: [current/default value]
param2 title (param2 name) [choice1, choice2, ...]: [current/default value]
...
```

This screen is usually the first interactive screen that is displayed when an administrative command is invoked interactively unless the invoked command does not contain any target object and non-step command parameters.If a command does not have a target object, then the prompt for the target object is skipped. The number of parameters depends on the number of arguments in the Argument area of the command-specific help. If an input is required, then an asterisk (*) is placed in front of the title. The parameter name is displayed for information and is the name that is used to set this parameter in batch mode. If a parameter value is restricted to a set of values, then the valid choices are displayed. If current or default value is available, it is displayed. You can accept the existing value by clicking Enter. To add or change an existing value, enter a new value and click Enter.

Display command execution menu

If an administrative command does not contain a step, you are presented with the following menu after collecting values for target object and parameters:

```
Command title

F (Finish)
C (Cancel)

Select [F, C]: F
```

The Finish option runs the command and the Cancel option cancels the command. The default selection is F (Finish). This menu is the last menu that is displayed for a non-step command to exit interactive mode by either canceling or running the command.

Display command step selection and execution menu

If an administrative command contains a step, the following menu is displayed after collecting values for target object and parameters:

```
Command title
Command description
 -> *1. step1 title (step1 name)
     2. step2 title (step2 name)
    *3. step3 title (step3 name)
    (4. step4 title (step4 name))
     ...
     n. stepn title (stepn name)

S (Select)
N (Next)
P (Previous)
F (Finish)
C (Cancel)
H (Help)

Select [S, N, P, F, C, H]: S
```

The number of steps that is displayed in the menu depends on the adminstative command. The step name is displayed for information and is the name that is used to set data in this step in batch mode. The following notations are used to describe a step:

- A "->" before the step indicates the current step position.
- A "*" before the step indicates a required step.
- A ( ) enclosing the entire step indicates a disabled step. You cannot navigate to this step by using the Next or Previous options.

Using the menu, you can navigate through steps sequentially by selecting Previous or Next. Select selects the current step, Finish runs the command, Cancel cancels the command, and Help provides on-line help for the command. Not all menu choices are available. Previous is not available if current step is the first step. Next is not be available if current step is the last step. Finish is not available if still steps are still missing required inputs. The default selection is S (Select) if the current step is a valid step and there are still steps missing required inputs. Default selection is F (Finish) if there is no missing required input in any step.

For commands with steps, this is the menu to exit interactive mode by either canceling or executing the command.

Display collection step menu

A step might or might not support collection. A collection refers to objects of the same type. In an administrative command, a collection contains objects with each having the same set of parameters. If a step supporting collection is selected, the wsadmin tool displays the following menu to add and select an object in the collection:

```
Step title (step name)
    | key param1 title (key param1 name), key param2 title (key param2 name), ...
    ---------------------------------------------------------------------------
->  | object1 key param1 value, key param2 value, ...
   *| object2 key param1 value, key param2 value, ...
    ...
key param1 title, key param2 title, ... must be provided to specify a row in batch row.

S (Select Row)
N (Next)
P (Previous)
A (Add Row or Add Row Before)
D (Delete Row)
F (Finish)
H (Help)

Select [S, N, P, A, D, F, H]: F
```

The number of objects that display in the menu depends on the command step. Key parameters are identified by the step to use to uniquely identify an object in a collection. Key parameter values are displayed so as to identify an object to select. As with the command step selection menu, an arrow (->) is used to indicate the current object position, and a asterisk (*) is used to indicate that required input is missing in the object.

Use the menu to navigate through objects sequentially by selecting Previous or Next. Select Row selects the current object, Add Row adds a new object, Add Row Before adds a new object before the current object, Delete Row deletes the current object, Finish returns control back to the step selection and execution menu, and Help provides on-line help for the step. Not all menu choices are available. Previous is not available if there is no object in the collection or the first object is the current object. Next is not available if there is no object in the collection or the last object is the current object. Select Row is available only if there is a current object. Add Row is provided only if there is no object in the collection and the step supports new object to be added. Add Row Before is provided if the step supports new object to be added and there are existing objects in the collection. Delete Row is provided only if there is a current object and the step allows object to be deleted. Finish is not available if there are still objects missing required inputs. Default selection is A (Add Row) when there is no object in the collection and the step supports objects to be added. Default selection is S (Select Row) if there is a current object and there are still objects missing required inputs. Default selection is F (Finish) if there is no required input missing in any object.

Collect user inputs for parameters of a collection object

After a collection object is selected, the parameter value for each parameter is prompted sequentially as shown in the following example:

```
*param1 title (param1 name) [choice1, choice2, ...]: [current/default value]
param2 title (param2 name) [choice1, choice2, ...]: [current/default value]
...
```

The number of parameters depends on the number of arguments in the “Argument” area of the command step specific help. The same “*” notation is used to denote required parameter. If a parameter value is restricted to a set of values, then the valid choices are displayed. If current or default value is available, it is displayed. For each writable parameter, you can accept the existing value by pressing the Enter key. To add or change an existing value, user simply enters a new value and then presses Enter. For a read-only parameter, the parameter and its value are displayed. However, user is not given the prompt to modify its value. Once user steps through all the parameters, wsadmin leads user back to the collection step menu.

Collect user inputs for non-collection step

There are two parts for this step. The first part is to display the current or default parameter values for the selected step as shown below:

```
Step title (step name)

*param1 title (param1 name) [choice1, choice2, ...]: [current/default value]
param2 title (param2 name) [choice1, choice2, ...]: [current/default value]
...

Select [C (Cancel), E (Edit)]: [E]
```

There is no prompting in this part. Instead, this part is more like a help providing parameter information on the selected step. The number of parameters depends on the number of arguments in the `argument` area of the command step specific help. The asterisk (*) notation denotes a required parameter. If a parameter value is restricted to a set of values, then the valid choices will be displayed. If current or default value is available, it is displayed. You can choose to cancel the step or continue to the next part to provide parameter inputs. The default selection is `Edit`. Since it is possible that you are seeing default values assigned to a new piece of data that is not yet set in the step, you should always accept the default selection to continue to the next part. Otherwise, if there is no data in the selected step, selecting Cancel will not result in creating the data.

If you accept the default `Edit` selection, collect user inputs for parameters sequentially just like “Collect user inputs for parameters of a collection object”.

```
*param1 title (param1 name) [choice1, choice2, ...]: [current/default value]
param2 title (param2 name) [choice1, choice2, ...]: [current/default value]
...
```

For each writable parameter, you can accept the existing value by clicking Enter. To add or change an existing value, enter a new value and then press Enter. For a read-only parameter, the parameter and its value are displayed. You will not be given the prompt to modify the value of the parameter. As soon as you step through all the parameters, the wsadmin tool will lead you back to the command step selection and execution menu.

Starting the wsadmin scripting client

The WebSphere Application Server wsadmin tool provides the ability to run scripts. You can use the wsadmin tool to manage a WebSphere Application Server V6.0 installation, as well as configuration, application deployment, and server run-time operations. The WebSphere Application Server only supports the Jacl and Jython scripting languages.

The wsadmin launcher makes several WebSphere Application Server scripting objects available: AdminConfig, AdminControl, AdminApp, AdminTask, and Help. Scripts use these objects for application management, configuration, operational control, and for communication with MBeans that run in WebSphere Application Server processes.

You must start the wsadmin scripting client before you perform any other task using scripting.

1. Locate the command that starts the wsadmin scripting client.

   The command for invoking a scripting process is located in the *install_root*/profiles/*profile_name*/bin directory. Use the `wsadmin.bat` file for a Windows system, and the `wsadmin.sh` file for a Linux or a UNIX system.

2. Start the wsadmin scripting client. You can start the wsadmin scripting client in several different ways. To specify the method for running scripts, perform one of the following wsadmin tool options:

Option for starting the wsadmin scripting client:	Explanation:	Examples:

Run scripting commands interactively	Run wsadmin with an option other than `-f` or `-c` or without an option.
	An interactive shell is displayed with a wsadmin prompt. From the wsadmin prompt, enter any Jacl or Jython command. You can also invoke commands using the AdminControl, AdminApp, AdminConfig, AdminTask, or Help wsadmin objects.
	To leave an interactive scripting session, use the **quit** or **exit** commands. These commands do not take any arguments.
		Using Jacl on Windows systems:
		`wsadmin.bat`
		Using Jacl on Unix systems:
		`wsadmin.sh`
		If security is enabled:
		`wsadmin.sh -user wsadmin -password wsadmin`
		Using Jython on Windows systems:
		`wsadmin.bat -lang jython`
		Using Jython on Unix systems:
		`wsadmin.sh -lang jython`
		By default security is enabled:
		`wsadmin.sh -lang jython -user wsadmin`
		`-password wsadmin`
		Example output:
		`WASX7209I: Connected to process server1 on`
		`node myhost using SOAP connector;`
		`The type of process is: UnManagedProcess`
		`WASX7029I: For help, enter: "$Help help"`
		`wsadmin>$AdminApp list`
		`adminconsole`
		`DefaultApplication`
		`ivtApp`
		`wsadmin>exit`

Run scripting commands as individual commands	Run the wsadmin tool with the -c option.	Using Jacl on Windows systems:
		`wsadmin -c "$AdminApp list"`
		Using Jacl on Unix systems:
		`wsadmin.sh -c "\$AdminApp list"`
		or
		`wsadmin.sh -c '$AdminApp list'`
		Using Jython on Windows systems:
		`wsadmin -lang jython`
		`-c "AdminApp.list()"`
		Using Jython on Linux or Unix systems:
		`wsadmin.sh -lang jython -c 'AdminApp.list()'`
		Example output:
		`WASX7209I: Connected to process "server1"`
		`on node myhost using SOAP connector;`
		`The type of process is: UnManagedProcess`
		`adminconsole`
		`DefaultApplication`
		`ivtApp`

Run scripting commands in a script	Run the wsadmin tool with the `-f` option, and place the commands that you want to run into the file.	Using Jacl on Windows systems:
		`wsadmin -f` *`al.jacl`*
		Using Jacl on Unix systems:
		`wsadmin.sh -f` *`al.jacl`*
		where the `al.jacl` file contains the following commands:
		`set apps [$AdminApp list]`
		`puts $apps`
		Using Jython on Windows systems:
		`wsadmin -lang jython -f` *`al.py`*
		Using Jython on Unix systems:
		`wsadmin.sh -lang jython -f` *`al.py`*
		where the `al.py` file contains the following commands:
		`apps = AdminApp.list()`
		`print apps`
		Example output:
		`WASX7209I: Connected to process "server1"`
		`on node myhost using SOAP connector;`
		`The  type of process is: UnManagedProcess`
		` adminconsole`
		` DefaultApplication`
		` ivtApp`

Run scripting commands in a profile script	A *profile script* is a script that runs before the main script, or before entering interactive mode. You can use profile scripts to set up a scripting environment that is customized for the user or the installation.
	To run scripting commands in a profile script, run the wsadmin tool with the -profile option, and include the commands that you want to run into the profile script.
	To customize the script environment, specify one or more profile scripts to run.
		Using Jacl on Windows systems:
		`wsadmin.bat -profile alprof.jacl`
		Using Jacl on Linux or Unix systems:
		`wsadmin.sh -profile alprof.jacl`
		where the `alprof.jacl` file contains the following commands:
		`set apps [$AdminApp list]`
		`puts "Applications currently`
		`installed:\n$apps"`
		Example output:
		`WASX7209I: Connected to process "server1"`
		`on node myhost using SOAP connector;`
		`The type of process is: UnManagedProcess`
		`Applications currently installed:`
		` adminconsole`
		` DefaultApplication`
		` ivtApp`
		` WASX7029I: For help, enter: "$Help help"`
		` wsadmin>`
		Using Jython on Windows systems:
		`wsadmin.bat -lang jython -profile alprof.py`
		Using Jython on Linux or Unix systems:
		`wsadmin.sh -lang jython -profile alprof.py`
		where the `alprof.py` file contains the following commands:
		`apps = AdminApp.list()`
		`print "Applications currently installed:\n "`
		`+ apps`
		Example output:
		`WASX7209I: Connected to process "server1" on`
		`node myhost using SOAP connector;`
		`The type of process is: UnManagedProcess`
		`Applications currently installed:`
		` adminconsole`
		` DefaultApplication`
		` ivtApp`
		` WASX7029I: For help, enter: "Help.help()"`
		` wsadmin>`

Scripting: Resources for learning

Use the following links to find relevant supplemental information about the Jacl and Jython scripting languages, and about using scripting with WebSphere Application Server. 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 instructions and examples

- Java command language
- Jacl: A Tcl implementation in Java

- Charming Jython
- Jython
- Sample scripts for WebSphere Application Server

Deploying applications using scripting

This topic contains the following tasks:

- Installing applications
- Uninstalling applications

Installing applications with the wsadmin tool

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

On a single server installation, the server must be running before you install an application. See the “Starting servers using scripting” on page 302 article for more information. On a network deployment installation, the deployment manager must be running before you install an application. See the “startManager command” on page 671article for more information.

You can install the application in batch mode, using the **install** command, or you can install the application in interactive mode using the **installinteractive** command. Interactive mode prompts you through a series of tasks to provide information. Both the **install** command and the **installinteractive** command support a set of options. See the “Options for the AdminApp object install, installInteractive, edit, editInteractive, update, and updateInteractive commands” on page 437 article for a list of valid options for the **install** and **installinteractive** commands. You can also obtain a list of supported options for an Enterprise Archive (EAR) file using the **options** command, for example:

Using Jacl:

```
$AdminApp options
```

Using Jython:

```
AdminApp.options()
```

For more information for the **options**, **install**, or **installinteractive** commands, see the “Commands for the AdminApp object” on page 415 article.

The application that you install must be an enterprise archive file (EAR), a Web archive (WAR) file, or a Java archive (JAR) file. The archive file must end in `.ear`, `.jar` or `.war` for the wsadmin tool to be able to install it. The wsadmin tool uses these extensions to figure out the archive type. If the file is a WAR or JAR file, it will be automatically wrapped as an EAR file.

If you are installing an application that has the AdminApp useMetaDataFromBinary option specified, then you can only install this application on a WebSphere Application Server V6.x deployment target. This also applies to editing the application, using the AdminApp **edit** command, after you install it. If you use the V5.x wsadmin tool to install or edit an application on a WebSphere Application Server V6.x cell, only the steps available for the V5.x wsadmin tool will be shown.

Perform the following steps to install an application into the run time:

1. Install the application.
   - Using batch mode:
     – For a single server installation only, the following example uses the EAR file and the command option information to install the application:
       - Using Jacl:

```
$AdminApp install c:/MyStuff/application1.ear {-server serv2}
```

- Using Jython list:

```
AdminApp.install('c:/MyStuff/application1.ear', ['-server', 'serv2'])
```

- Using Jython string:

```
AdminApp.install('c:/MyStuff/application1.ear', '[-server serv2]')
```

where:

$	is a Jacl operator for substituting a variable name with its value
AdminApp	is an object supporting application object management
install	is an AdminApp command
MyStuff/application1.ear	is the name of the application to install
server	is an installation option
serv2	is the value of the server option

– For a network deployment installation only, the following command uses the EAR file and the command option information to install the application on a cluster:

- Using Jacl:

```
$AdminApp install c:/MyStuff/application1.ear {-cluster cluster1}
```

- Using Jython list:

```
AdminApp.install('c:/MyStuff/application1.ear', ['-cluster', 'cluster1'])
```

- Using Jython string:

```
AdminApp.install('c:/MyStuff/application1.ear', '[-cluster cluster1]')
```

where:

$	is a Jacl operator for substituting a variable name with its value
AdminApp	is an object allowing application objects to be managed
install	is an AdminApp command
MyStuff/application1.ear	is the name of the application to install
cluster	is an installation option
cluster1	the value of the cluster option which will be cluster name

- Using interactive mode, the following command changes the application information by prompting you through a series of installation tasks:

– Using Jacl:

```
$AdminApp installInteractive c:/MyStuff/application1.ear
```

– Using Jython:

```
AdminApp.installInteractive('c:/MyStuff/application1.ear')
```

where:

$	is a Jacl operator for substituting a variable name with its value
AdminApp	is an object allowing application objects to be managed
installInteractive	is an AdminApp command
MyStuff/application1.ear	is the name of the application to install

2. Save the configuration changes. See the “Saving configuration changes with the wsadmin tool” on page 232 article for more information.

3. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 article for more information.

Uninstalling applications with the wsadmin tool

Before starting this task, the wsadmin tool must be running. See “Starting the wsadmin scripting client” on page 250 for more information.

Steps to uninstall an application follow:

1. Uninstall the application:

   Specify the name of the application you want to uninstall, not the name of the Enterprise ARchive (EAR) file.

   - Using Jacl:

     ```
     $AdminApp uninstall application1
     ```

   - Using Jython:

     ```
     AdminApp.uninstall('application1')
     ```

   where:

`$`	is a Jacl operator for substituting a variable name with its value
`AdminApp`	is an object supporting application objects management
`uninstall`	is an AdminApp command
`application1`	is the name of the application to uninstall

2. Save the configuration changes. See the “Saving configuration changes with the wsadmin tool” on page 232 article for more information.
3. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 article for more information.

Uninstalling an application removes it from the WebSphere Application Server configuration and from all the servers that the application was installed on. The application binaries (EAR file contents) are deleted from the installation directory. This occurs when the configuration is saved for single server WebSphere Application Server editions or when the configuration changes are synchronized from deployment manager to the individual nodes for network deployment configurations.

Managing deployed applications using scripting

This topic contains the following tasks:

- “Starting applications with scripting” on page 257
- “Updating installed applications with the wsadmin tool” on page 258
- “Stopping applications with scripting” on page 261
- “Listing the modules in an installed application with scripting” on page 262
- “Querying application state using scripting” on page 266
- “Disabling application loading in deployed targets using scripting” on page 267
- “Configuring applications for session management using scripting” on page 268
- “Configuring applications for session management in Web modules using scripting” on page 271
- “Exporting applications using scripting” on page 275
- “Configuring a shared library using scripting” on page 275
- “Configuring a shared library for an application using scripting” on page 279
- “Setting background applications using scripting” on page 282

Starting applications with scripting

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

Before starting an application, it must be installed. See the “Installing applications with the wsadmin tool” on page 254 article for more information.

Perform the following steps to start an application:

1. Identify the application manager MBean for the server where the application resides and assign it the appManager variable. The following example returns the name of the application manager MBean.
   - Using Jacl:
     ```
     set appManager [$AdminControl queryNames cell=mycell,node=mynode,type=ApplicationManager,
     process=server1,*]
     ```
   - Using Jython:
     ```
     appManager = AdminControl.queryNames('cell=mycell,node=mynode,type=ApplicationManager,
      process=server1,*')
     print appManager
     ```

   where:

set	is a Jacl command
appManager	is a variable name
$	is a Jacl operator for substituting a variable name with its value
AdminControl	is an object that enables the manipulation of MBeans running in a WebSphere server process
queryNames	is an AdminControl command

cell=*mycell*,node=*mynode*,type=*ApplicationManager*,
process=*server1*
	is the hierarchical containment path of the configuration object

print	is a Jython command

   Example output:

   ```
   WebSphere:cell=mycell,name=ApplicationManager,mbeanIdentifier=ApplicationManager,type=ApplicationManager,
    node=mynode,process=server1
   ```

2. Start the application. The following example invokes the startApplication operation on the MBean, providing the application name that you want to start.
   - Using Jacl:
     ```
     $AdminControl invoke $appManager startApplication myApplication
     ```
   - Using Jython:
     ```
     AdminControl.invoke(appManager, 'startApplication', 'myApplication')
     ```

   where:

$	is a Jacl operator for substituting a variable name with its value
AdminControl	is an object that enables the manipulation of MBeans running in a WebSphere server process
invoke	is an AdminControl command
appManager	evaluates to the ID of the server specified in step number 1
startApplication	is an attribute of modify objects

myApplication	is the value of the startApplication attribute

Updating installed applications with the wsadmin tool

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

Before starting an application, it must be installed. See the “Installing applications with the wsadmin tool” on page 254 article for more information.

Both the **update** command and the **updateinteractive** command support a set of options. See the “Options for the AdminApp object install, installInteractive, edit, editInteractive, update, and updateInteractive commands” on page 437 article for a list of valid options for the **update** and **updateinteractive** commands. You can also obtain a list of supported options for an Enterprise Archive (EAR) file using the **options** command, for example:

Using Jacl:

```
$AdminApp options
```

Using Jython:

```
print AdminApp.options()
```

For more information for the **options**, **update**, or **updateinteractive** commands, see the “Commands for the AdminApp object” on page 415 article. Perform the following steps to update an application:

1. Update the installed application using one of the following options:
   - The following command updates a single file in a deployed application:
     – Using Jacl:

       ```
       $AdminApp update app1 file {-operation update -contents c:/apps/app1/my.xml
        -contenturi app1.jar/my.xml}
       ```

     – Using Jython string:

       ```
       AdminApp.update('app1', 'file', '[-operation update -contents c:/apps/app1/my.xml
        -contenturi app1.jar/my.xml]')
       ```

     – Using Jython list:

       ```
       AdminApp.update('app1', 'file', ['-operation', 'update', '-contents', 'c:/apps/app1/my.xml',
        '-contenturi', 'app1.jar/my.xml'])
       ```

     where:

$	is a Jacl operator for substituting a variable name with its value
AdminApp	is an object supporting application objects management
update	is an AdminApp command
app1	is the name of the application to update
file	is the content type value
operation	is an option of the **update** command
update	is the value of the `operation` option
contents	is an option of the **update** command
/apps/app1/my.xml	is the value of the `contents` option
contenturi	is an option of the **update** command
app1.jar/my.xml	is the value of the `contenturi` option

- The following command adds a module to the deployed application, if the module does not exist. Otherwise, the existing module will be updated.
  - Using Jacl:

    ```
    $AdminApp update app1 modulefile {-operation addupdate -contents
    c:/apps/app1/Increment.jar -contenturi Increment.jar -nodeployejb
    -BindJndiForEJBNonMessageBinding {{"Increment Enterprise Java Bean"
     Increment Increment.jar,META-INF/ejb-jar.xml Inc}}}
    ```

  - Using Jython string:

    ```
    AdminApp.update('app1', 'modulefile', '[-operation addupdate -contents
    c:/apps/app1/Increment.jar -contenturi Increment.jar -nodeployejb
    -BindJndiForEJBNonMessageBinding [["Increment Enterprise Java Bean
    " Increment Increment.jar,META-INF/ejb-jar.xml Inc]]]')
    ```

  - Using Jython list:

    ```
    bindJndiForEJBValue = [["Increment Enterprise Java Bean",
    "Increment", " Increment.jar,
     META-INF/ejb-jar.xml", "Inc"]]

    AdminApp.update('app1', 'modulefile', ['-operation', 'addupdate', '-contents',
    'c:/apps/app1/Increment.jar', '-contenturi','Increment.jar' '-nodeployejb',
    `-BindJndiForEJBNonMessageBinding', bindJndiForEJBValue])
    ```

  where:

$	is a Jacl operator for substituting a variable name with its value
AdminApp	is an object supporting application objects management
update	is an AdminApp command
app1	is the name of the application to update
modulefile	is the content type value
operation	is an option of the **update** command
addupdate	is the value of the operation option
contents	is an option of the **update** command
/apps/app1/Increment.jar	is the value of the contents option
contenturi	is an option of the **update** command
Increment.jar	is the value of the contenturi option
nodeployejb	is an option of the **update** command
BindJndiForEJBNonMessageBinding	is an option of the **update** command
″Increment Enterprise Java Bean″ Increment Increment.jar,META-INF/ejb-jar.xml Inc	is the value of the BindJndiForEJBNonMessageBinding option
bindJndiForEJBValue	is a Jython variable containing the value of the BindJndiForEJBNonMessageBinding option

- The following command uses a partial application to update a deployed application:
  - Using Jacl:

    ```
    $AdminApp update app1 partialapp {-contents c:/apps/app1/app1Partial.zip}
    ```

  - Using Jython string:

    ```
    AdminApp.update('app1', 'partialapp', '[-contents c:/apps/app1/app1Partial.zip]')
    ```

  - Using Jython list:

    ```
    AdminApp.update('app1', 'partialapp', ['-contents', 'c:/apps/app1/app1Partial.zip'])
    ```

where:

$	is a Jacl operator for substituting a variable name with its value
AdminApp	is an object supporting application objects management
update	is an AdminApp command
app1	is the name of the application to update
partialapp	is the content type value
contents	is an option of the **update** command
/apps/app1/app1Partial.zip	is the value of the contents option

- The following command updates the entire deployed application:
  - Using Jacl:

    ```
    $AdminApp update app1 app {-operation update -contents c:/apps/app1/newApp1.jar
    -usedefaultbindings -nodeployejb -BindJndiForEJBNonMessageBinding
    {{"Increment Enterprise Java Bean" Increment Increment.jar,META-INF/ejb-jar.xml Inc}}}
    ```
  - Using Jython string:

    ```
    AdminApp.update('app1', 'app', '[-operation update -contents c:/apps/app1/newApp1.ear
    -usedefaultbindings -nodeployejb -BindJndiForEJBNonMessageBinding
    [["Increment Enterprise Java Bean" Increment Increment.jar,META-INF/ejb-jar.xml Inc]]]')
    ```
  - Using Jython list:

    ```
    bindJndiForEJBValue = [["Increment Enterprise Java Bean", "Increment", " Increment.jar,
    META-INF/ejb-jar.xml", "Inc"]]

    AdminApp.update('app1', 'app', ['-operation', 'update', '-contents',
    'c:/apps/app1/NewApp1.ear', '-usedefaultbindings', '-nodeployejb',
    `-BindJndiForEJBNonMessageBinding', bindJndiForEJBValue])
    ```

where:

$	is a Jacl operator for substituting a variable name with its value
AdminApp	is an object supporting application objects management
update	is an AdminApp command
app1	is the name of the application to update
app	is the content type value
operation	is an option of the **update** command
update	is the value of the operation option
contents	is an option of the **update** command
/apps/app1/newApp1.ear	is the value of the contents option
usedefaultbindings	is an option of the **update** command
nodeployejb	is an option of the **update** command
BindJndiForEJBNonMessageBinding	is an option of the **update** command
″Increment Enterprise Java Bean″ Increment Increment.jar,META-INF/ejb-jar.xml Inc	is the value of the BindJndiForEJBNonMessageBinding option
bindJndiForEJBValue	is a Jython variable containing the value of the BindJndiForEJBNonMessageBinding option

2. Save the configuration changes. See the “Saving configuration changes with the wsadmin tool” on page 232 article for more information.

3. In a network deployment environment only, synchronize the node. See the "Synchronizing nodes with the wsadmin tool" on page 217 article for more information.

Stopping applications with scripting

Before starting this task, the wsadmin tool must be running. See the "Starting the wsadmin scripting client" on page 250 article for more information.

The following example stops all running applications on a server:

1. Identify the application manager MBean for the server where the application resides, and assign it to the appManager variable.
   - Using Jacl:

     ```
     set appManager [$AdminControl queryNames cell=mycell,node=mynode,type=ApplicationManager,
      process=server1,*]
     ```

   - Using Jython:

     ```
     appManager = AdminControl.queryNames('cell=mycell, node=mynode,type=ApplicationManager,
      process=server1,*')
     print appManager
     ```

   where:

set	is a Jacl command
appManager	is a variable name
$	is a Jacl operator for substituting a variable name with its value
AdminControl	is an object that enables the manipulation of MBeans running in a WebSphere server process
queryNames	is an AdminControl command

cell=*mycell*,node=*mynode*,type=*ApplicationManager*,
process=*server1*
	is the hierarchical containment path of the configuration object

print	is a Jython command

   This command returns the application manager MBean.

   Example output:

   ```
   WebSphere:cell=mycell,name=ApplicationManager,mbeanIdentifier=ApplicationManager,type=ApplicationManager,
    node=mynode,process=server1
   ```

2. Query the running applications belonging to this server and assign the result to the apps variable.
   - Using Jacl:

     ```
     set apps [$AdminControl queryNames cell=mycell,node=mynode,type=Application,process=server1,*]
     ```

   - Using Jython:

     ```
     # get line separator
     import  java.lang.System  as sys
     lineSeparator = sys.getProperty('line.separator')

     apps = AdminControl.queryNames('cell=mycell,node=mynode,type=Application,process=server1,*').split(lineSeparator)
     print apps
     ```

   where:

set	is a Jacl command
apps	is a variable name
$	is a Jacl operator for substituting a variable name with its value

AdminControl	is an object that enables the manipulation of MBeans running in a WebSphere server process
queryNames	is an AdminControl command

cell=*mycell*,node=*mynode*,type=*ApplicationManager*,
process=*server1*
	is the hierarchical containment path of the configuration object

print	is a Jython command

This command returns a list of application MBeans.

Example output:

```
WebSphere:cell=mycell,name=adminconsole,mbeanIdentifier=deployment.xml#ApplicationDeployment_1,
type=Application,node=mynode,Server=server1,process=server1,J2EEName=adminconsole
WebSphere:cell=mycell,name=filetransfer,mbeanIdentifier=deployment.xml#ApplicationDeployment_1,
type=Application,node=mynode,Server=server1,process=server1,J2EEName=filetransfer
```

3. Stop all the running applications.
   - Using Jacl:

     ```
     foreach app $apps {
          set appName [$AdminControl getAttribute $app name]
          $AdminControl invoke $appManager stopApplication $appName}
     ```
   - Using Jython:

     ```
     for app in apps:
         appName = AdminControl.getAttribute(app, 'name')
         AdminControl.invoke(appManager, 'stopApplication', appName)
     ```

   This command stops all the running applications by invoking the stopApplication operation on the MBean, passing in the application name to stop.

Once you complete the steps for this task, all running applications on the server are stopped.

Listing the modules in an installed application with scripting

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

Use the AdminApp object **listModules** command to list the modules in an installed application. For example:

- Using Jacl:

  ```
  $AdminApp listModules DefaultApplication -server
  ```
- Using Jython:

  ```
  print AdminApp.listModules('DefaultApplication', '-server')
  ```

where:

$	is a Jacl operator for substituting a variable name with its value
print	is a Jython command
AdminApp	is an object supporting application object management
listmodules	is an AdminApp command
DefaultApplication	is the name of the application
-server	is an optional option specified

Example output:

```
DefaultApplication#IncCMP11.jar+META-INF/ejb-jar.xml#WebSphere:cell=mycell,node=mynode,server=myserver
DefaultApplication#DefaultWebApplication.war+WEB-INF/web.xml#WebSphere:cell=mycell,node=mynode,server=myserver
```

Example: Listing the modules in an application server: The following example lists all modules on all enterprise applications installed on `server1` in `node1`:

Note: ***** means that the module is installed on server1 node node1 and other node and/or server.

+ means that the module is installed on server1 node node1 only means that the module is not installed on server1 node node1.

```
 1  #----------------------------------------------------------------------------
       2  # setting up variables to keep server name and node name
       3  #------------------------------------------------------------------------
       4  set  serverName  server1
       5  set  nodeName  node1
       6  #----------------------------------------------------------------------------
       7  # setting up 2 global lists to keep the modules
       8  #----------------------------------------------------------------------------
       9  set  ejbList {}
      10 set webList {}
11
12 #----------------------------------------------------------------------------------
13 # gets all deployment objects and assigned it to deployments variable
14 #----------------------------------------------------------------------------------
15 set deployments [$AdminConfig getid /Deployment:/]
16
17 #---------------------------------------------------------------------------------------
18 # lines 22 thru 148 Iterates through all the deployment objects to get the modules
19 # and perform filtering to list application that has at least one module installed
20 # in server1 in node myNode
21 #---------------------------------------------------------------------------------------
22 foreach deployment $deployments {
23
24     # -------------------------------------------------------------------------------
25     # reset the lists that hold modules for each application
26     #--------------------------------------------------------------------------------
27     set webList {}
28     set ejbList {}
29
30     #-----------------------------------------
31     # get the application name
32     #-----------------------------------------
33     set appName [lindex [split $deployment (] 0]
34
35     #-----------------------------------------
36     # get the deployedObjects
37     #-----------------------------------------
38     set depObject [$AdminConfig showAttribute $deployment deployedObject]
39
40     #-------------------------------------------
41     # get all modules in the application
42     #--------------------------------------------
43     set modules [lindex [$AdminConfig showAttribute $depObject modules] 0]
44
45     #---------------------------------------------------------------------------------------------
46     # initialize lists to save all the modules in the appropriate list to where they belong
47     #---------------------------------------------------------------------------------------------
48     set modServerMatch {}
49     set modServerMoreMatch {}
50     set modServerNotMatch {}
51
52         #-----------------------------------------------------------------------------------------
53         # lines 55 to 112 iterate through all modules to get the targetMappings
54         #-----------------------------------------------------------------------------------------
55         foreach module $modules {
```

```
56            #-----------------------------------------------------------------------------------------
57            # setting up some flag to do some filtering and get modules for server1 on node1
58            #-----------------------------------------------------------------------------------------
59            set sameNodeSameServer "false"
60           set diffNodeSameServer "false"
61            set sameNodeDiffServer "false"
62            set diffNodeDiffServer "false"
63
64            #---------------------------------------------
65            # get the targetMappings
66            #---------------------------------------------
67            set targetMaps [lindex [$AdminConfig showAttribute $module targetMappings] 0]
68
69            #--------------------------------------------------------------------------------------------
70           # lines 72 to 111 iterate through all targetMappings to get the target
71            #--------------------------------------------------------------------------------------------
72            foreach targetMap $targetMaps {
73                #-------------------------------
74               # get the target
75                #-------------------------------
76                set target [$AdminConfig showAttribute $targetMap target]
77
78                #---------------------------------------------------
79                # do filtering to skip ClusteredTargets
80                #---------------------------------------------------
81                set targetName [lindex [split $target #] 1]
82               if {[regexp "ClusteredTarget" $targetName] != 1} {
83                    set sName [$AdminConfig showAttribute $target name]
84                    set nName [$AdminConfig showAttribute $target nodeName]
85
86                    #-----------------------------------------------
87                    # do the server name match
88                    #-----------------------------------------------
89                    if {$sName == $serverName} {
90                      if {$nName == $nodeName} {
91                set sameNodeSameServer "true"
92                        } else {
93                set diffNodeSameServer "true"
94                        }
95                   } else {
96                         #----------------------------------------
97                         # do the node name match
98                         #----------------------------------------
99                         if {$nName == $nodeName} {
100                set sameNodeDiffServer "true"
101                        } else {
102                            set diffNodeDiffServer "true"
103                        }
104                 }
105
106                 if {$sameNodeSameServer == "true"} {
107                     if {$sameNodeDiffServer == "true" || $diffNodeDiffServer == "true" ||
            $diffNodeSameServer == "true"} {
108                         break
109                     }
110                }
111           }
112     }
113
114     #----------------------------------------------
115     # put it in the appropriate list
116     #----------------------------------------------
117     if {$sameNodeSameServer == "true"} {
118         if {$diffNodeDiffServer == "true" || $diffNodeSameServer == "true" || $sameNodeDiffServer == "true"}
       {
119              set modServerMoreMatch [linsert $modServerMoreMatch end [$AdminConfig showAttribute
         $module uri]]
         } else {
             set modServerMatch [linsert $modServerMatch end [$AdminConfig showAttribute $module uri]]
        }
   } else {
        set modServerNotMatch [linsert $modServerNotMatch end [$AdminConfig showAttribute $module uri]]
   }
}


#------------------------------------------------------------------
# print the output with some notation as a mark
#------------------------------------------------------------------
if {$modServerMatch != {} || $modServerMoreMatch != {}} {
    puts stdout "\tApplication name: $appName"
}

#------------------------------------------------------------
# do grouping to appropriate module and print
#------------------------------------------------------------
if {$modServerMatch != {}} {
    filterAndPrint $modServerMatch "+"
}
if {$modServerMoreMatch != {}} {
    filterAndPrint $modServerMoreMatch "*"
}
if {($modServerMatch != {} || $modServerMoreMatch != {}) "" $modServerNotMatch != {}} {
    filterAndPrint $modServerNotMatch ""
}
}


proc filterAndPrint {lists flag} {
   global webList
   global ejbList
   set webExists "false"
   set ejbExists "false"

   #------------------------------------------------------------------------------------------
   # If list already exists, flag it so as not to print the title more then once
   # and reset the list
   #------------------------------------------------------------------------------------------
   if {$webList != {}} {
       set webExists "true"
       set webList {}
   }
   if {$ejbList != {}} {
       set ejbExists "true"
       set ejbList {}
   }

   #--------------------------------------------------------------------
   # do some filtering for web modules and ejb modules
   #--------------------------------------------------------------------
   foreach list $lists {
        set temp [lindex [split $list .] 1]
        if {$temp == "war"} {
             set webList [linsert $webList end $list]
        } elseif {$temp == "jar"} {
             set ejbList [linsert $ejbList end $list]
        }
   }

   #--------------------------------------
   # sort the list before printing
   #--------------------------------------
   set webList [lsort -dictionary $webList]
   set ejbList [lsort -dictionary $ejbList]
```

```

    #-----------------------------------------------------------------
    # print out all the web modules installed in server1
    #-----------------------------------------------------------------
    if {$webList != {}} {
        if {$webExists == "false"} {
             puts stdout "\t\tWeb Modules:"
        }
        foreach web $webList {
            puts stdout "\t\t\t$web  $flag"
        }
    }

    #---------------------------------------------------------------
    # print out all the ejb modules installed in server1
    #---------------------------------------------------------------
    if {$ejbList != {}} {
         if {$ejbExists == "false"} {
               puts stdout "\t\tEJB Modules:"
         }
         foreach ejb $ejbList {
               puts stdout "\t\t\t$ejb  $flag"
         }
    }
}
```

```
Example output for server1 on node node1:
       Application name: TEST1
                EJB Modules:
                        deplmtest.jar  +
                Web Modules:
                        mtcomps.war  *
        Application name: TEST2
                Web Modules:
                        mtcomps.war  +
                EJB Modules:
                        deplmtest.jar  +
        Application name: TEST3
                Web Modules:
                        mtcomps.war  *
                EJB Modules:
                        deplmtest.jar  *
        Application name: TEST4
                EJB Modules:
                        deplmtest.jar  *
                Web Modules:
                        mtcomps.war
```

Querying application state using scripting

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

The following example queries for the presence of Application MBean to find out whether the application is running.

Using Jacl:

```
$AdminControl completeObjectName type=Application,name=myApplication,*
```

Using Jython:

```
print AdminControl.completeObjectName('type=Application,name=myApplication,*')
```

where:

$	is a Jacl operator for substituting a variable name with its value
AdminControl	is an object that enables the manipulation of MBeans running in a WebSphere server process
completeObjectName	is an AdminControl command
type=Application,name=*myApplication*	is the hierarchical containment path of the configuration object
print	is a Jython command

If *myApplication* is running, then there should be an MBean created for it. Otherwise, the command returns nothing. If *myApplication* is running, the following is the example output:

```
WebSphere:cell=mycell,name=myApplication,mbeanIdentifier=cells/mycell/applications/myApplication.ear/deployments/myApplic
```

Disabling application loading in deployed targets using scripting

Before starting this task, the wsadmin tool must be running. See the "Starting the wsadmin scripting client" on page 250 article for more information.

The following example uses the AdminConfig object to disable application loading in deployed targets:

1. Obtain the deployment object for the application and assign it to the `deployments` variable, for example:
   - Using Jacl:
     ```
     set deployments [$AdminConfig getid /Deployment:myApp/]
     ```
   - Using Jython:
     ```
     deployments = AdminConfig.getid("/Deployment:myApp/")
     ```

   where:

set	is a Jacl command
deployments	is a variable name
$	is a Jacl operator for substituting a variable name with its value
AdminConfig	is an object representing the WebSphere Application Server configuration
getid	is an AdminConfig command
Deployment	is an attribute
myApp	is the value of the attribute

   Example output:
   ```
   myApp(cells/mycell/applications/myApp.ear/deployments/myApp|deployment.xml#Deployment_1)
   ```
2. Obtain the target mappings in the application and assign them to the `targetMappings` variable, for example:
   - Using Jacl:
     ```
     set deploymentObj1 [$AdminConfig showAttribute $deployments deployedObject]
     set targetMap1 [lindex [$AdminConfig showAttribute $deploymentObj1 targetMappings] 0]
     ```
     Example output:
     ```
     (cells/mycell/applications/ivtApp.ear/deployments/ivtApp|deployment.xml#DeploymentTargetMapping_1)
     ```
   - Using Jython:

```
deploymentObj1 = AdminConfig.showAttribute(deployments, 'deployedObject')
targetMap1 = AdminConfig.showAttribute(deploymentObj1, 'targetMappings')
targetMap1 = targetMap1[1:len(targetMap1)-1].split(" ")
print targetMap1
```

Example output:

```
['(cells/mycell/applications/ivtApp.ear/deployments/ivtApp|deployment.xml#DeploymentTargetMapping_1)']
```

where:

set	is a Jacl command
deploymentObj1	is a variable name
$	is a Jacl operator for substituting a variable name with its value
AdminConfig	is an object representing the WebSphere Application Server configuration
showAttribute	is an AdminConfig command
deployments	evaluates to the ID of the deployment object specified in step number 1
deployedObject	is an attribute
targetMap1	a variable name
targetMappings	is an attribute
lindex	a Jacl command
print	a Jython command

3. Disable the loading of the application on each deployed target, for example:
   - Using Jacl:
     ```
     foreach tm $targetMap1 {
           $AdminConfig modify $tm {{enable false}}
     }
     ```
   - Using Jython:
     ```
     for targetMapping in targetMap1:
        AdminConfig.modify(targetMapping, [["enable", "false"]])
     ```
4. Save the configuration changes. See the “Saving configuration changes with the wsadmin tool” on page 232 article for more information.
5. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 article for more information.

Configuring applications for session management using scripting

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

You can use the AdminApp object to set configurations in an application. Some configuration settings are not available through the AdminApp object. The following example uses the AdminConfig object to configure session manager for the application.

1. Identify the deployment configuration object for the application and assign it to the deployment variable. For example:
   - Using Jacl:
     ```
     set deployments [$AdminConfig getid /Deployment:myApp/]
     ```
   - Using Jython:
     ```
     deployments = AdminConfig.getid('/Deployment:myApp/')
     print deployment
     ```

where:

set	is a Jacl command
deployments	is a variable name
$	is a Jacl operator for substituting a variable name with its value
AdminConfig	is an object representing the WebSphere Application Server configuration
getid	is an AdminConfig command
Deployment	is an attribute
myApp	is the value of the attribute

Example output:

```
myApp(cells/mycell/applications/myApp.ear/deployments/myApp|deployment.xml#Deployment_1)
```

2. Retrieve the applicaton deployment and assign it to the appDeploy variable. For example:
   - Using Jacl:
     ```
     set appDeploy [$AdminConfig showAttribute $deployments deployedObject]
     ```
   - Using Jython:
     ```
     appDeploy = AdminConfig.showAttribute(deployments, 'deployedObject')
     print appDeploy
     ```

   where:

set	is a Jacl command
appDeploy	is a variable name
$	is a Jacl operator for substituting a variable name with its value
AdminConfig	is an object representing the WebSphere Application Server configuration
showAttribute	is an AdminConfig command
deployments	evaluates to the ID of the deployment object specified in step number 1
deployedObject	is an attribute

Example output:

```
(cells/mycell/applications/myApp.ear/deployments/myApp|deployment.xml#ApplicationDeployment_1)
```

3. To obtain a list of attributes you can set for session manager, use the **attributes** command. For example:
   - Using Jacl:
     ```
     $AdminConfig attributes SessionManager
     ```
   - Using Jython:
     ```
     print AdminConfig.attributes('SessionManager')
     ```

   where:

$	is a Jacl operator for substituting a variable name with its value
AdminConfig	is an object representing the WebSphere Application Server configuration
attributes	is an AdminConfig command
SessionManager	is an attribute

Example output:

```
"accessSessionOnTimeout Boolean"
"allowSerializedSessionAccess Boolean"
"context ServiceContext@"
"defaultCookieSettings Cookie"
"enable Boolean"
"enableCookies Boolean"
"enableProtocolSwitchRewriting Boolean"
"enableSSLTracking Boolean"
"enableSecurityIntegration Boolean"
"enableUrlRewriting Boolean"
"maxWaitTime Integer"
"properties Property(TypedProperty)*"
"sessionDRSPersistence DRSSettings"
"sessionDatabasePersistence SessionDatabasePersistence"
"sessionPersistenceMode ENUM(DATABASE, DATA_REPLICATION, NONE)"
"tuningParams TuningParams"
```

4. Set up the attributes for the session manager. The following example sets three top level attributes in the session manager. You can modify the example to set other attributes of session manager including the nested attributes in Cookie, DRSSettings, SessionDataPersistence, and TuningParms object types. To list the attributes for those object types, use the **attributes** command of the AdminConfig object.

   - Using Jacl:

     ```
     set attr1 [list enableSecurityIntegration true]
     set attr2 [list maxWaitTime 30]
     set attr3 [list sessionPersistenceMode NONE]
     set attrs [list $attr1 $attr2 $attr3]
     set sessionMgr [list sessionManagement $attrs]
     ```

     Example output using Jacl:

     ```
     sessionManagement {{enableSecurityIntegration true} {maxWaitTime 30} {sessionPersistenceMode NONE}}
     ```

   - Using Jython:

     ```
     attr1 = ['enableSecurityIntegration', 'true']
     attr2 = ['maxWaitTime', 30]
     attr3 = ['sessionPersistenceMode', 'NONE']
     attrs = [attr1, attr2, attr3]
     sessionMgr = [['sessionManagement', attrs]]
     ```

     Example output using Jython:

     ```
     [[sessionManagement, [[enableSecurityIntegration, true], [maxWaitTime, 30], [sessionPersistenceMode, NONE]]]
     ```

   where:

set	is a Jacl command
attr1, attr2, attr3, attrs, sessionMgr	are variable names
$	is a Jacl operator for substituting a variable name with its value
enableSecurityIntegration	is an attribute
true	is a value of the enableSecurityIntegration attribute
maxWaitTime	is an attribute
30	is a value of the maxWaitTime attribute
sessionPersistenceMode	is an attribute
NONE	is a value of the sessionPersistenceMode attribute

5. Create the session manager for the application. For example:

   - Using Jacl:

     ```
     $AdminConfig create ApplicationConfig $appDeploy [list $sessionMgr]
     ```

- Using Jython:

  ```
  print AdminConfig.create('ApplicationConfig', appDeploy, sessionMgr)
  ```

  where:

`$`	is a Jacl operator for substituting a variable name with its value
`AdminConfig`	is an object representing the WebSphere Application Server configuration
`create`	is an AdminConfig command
`ApplicationConfig`	is an attribute
`appDeploy`	evaluates to the ID of the deployed application specified in step number 2
`list`	is a Jacl command
`sessionMgr`	evaluates to the ID of the session manager specified in step number 4

Example output:

```
(cells/mycell/applications/myApp.ear/deployments/myApp|deployment.xml#ApplicationConfig_1)
```

6. Save the configuration changes. See the "Saving configuration changes with the wsadmin tool" on page 232 article for more information.
7. In a network deployment environment only, synchronize the node. See the "Synchronizing nodes with the wsadmin tool" on page 217 article for more information.

Configuring applications for session management in Web modules using scripting

Before starting this task, the wsadmin tool must be running. See the "Starting the wsadmin scripting client" on page 250 article for more information.

You can use the AdminApp object to set configurations in an application. Some configuration settings are not available through the AdminApp object. This example uses the AdminConfig object to configure session manager for Web module in the application.

1. Identify the deployment configuration object for the application and assign it to the deployment variable. For example:
   - Using Jacl:

     ```
     set deployments [$AdminConfig getid /Deployment:myApp/]
     ```

   - Using Jython:

     ```
     deployments = AdminConfig.getid('/Deployment:myApp/')
     print deployments
     ```

   where:

`set`	is a Jacl command
`deployments`	is a variable name
`$`	is a Jacl operator for substituting a variable name with its value
`AdminConfig`	is an object representing the WebSphere Application Server configuration
`getid`	is an AdminConfig command
`Deployment`	is an attribute
`myApp`	is the value of the attribute

Example output:

```
myApp(cells/mycell/applications/myApp.ear/deployments/myApp|deployment.xml#Deployment_1)
```

2. Get all the modules in the application and assign it to the modules variable. For example:
   - Using Jacl:

     ```
     set appDeploy [$AdminConfig showAttribute $deployments deployedObject]
     set mod1 [$AdminConfig showAttribute $appDeploy modules]
     ```

     Example output:

     ```
     (cells/mycell/applications/myApp.ear/deployments/myApp:deployment.xml#WebModuleDeployment_1)
     (cells/mycell/applications/myApp.ear/deployments/myApp:deployment.xml#EJBModuleDeployment_1)
     (cells/mycell/applications/myApp.ear/deployments/myApp:deployment.xml#WebModuleDeployment_2)
     ```
   - Using Jython:

     ```
     appDeploy = AdminConfig.showAttribute(deployments, 'deployedObject')
     mod1 = AdminConfig.showAttribute(appDeploy, 'modules')
     print mod1
     ```

     Example output:

     ```
     [(cells/mycell/applications/myApp.ear/deployments/myApp|deployment.xml#WebModuleDeployment_1)
     (cells/mycell/applications/myApp.ear/deployments/myApp|deployment.xml#EJBModuleDeployment_1)
     (cells/mycell/applications/myApp.ear/deployments/myApp|deployment.xml#EJBModuleDeployment_2)]
     ```

   where:

set	is a Jacl command
appDeploy	is a variable name
mod1	is a variable name
$	is a Jacl operator for substituting a variable name with its value
AdminConfig	is an object representing the WebSphere Application Server configuration
showAttribute	is an AdminConfig command
deployments	evaluates to the ID of the deployment object specified in step number 1
deployedObject	is an attribute

3. To obtain a list of attributes you can set for session manager, use the attributes command. For example:
   - Using Jacl:

     ```
     $AdminConfig attributes SessionManager
     ```
   - Using Jython:

     ```
     print AdminConfig.attributes('SessionManager')
     ```

   where:

$	is a Jacl operator for substituting a variable name with its value
AdminConfig	is an object representing the WebSphere Application Server configuration
attributes	is an AdminConfig command
SessionManager	is an attribute

   Example output:

```
"accessSessionOnTimeout Boolean"
"allowSerializedSessionAccess Boolean"
"context ServiceContext@"
"defaultCookieSettings Cookie"
"enable Boolean"
```

```
"enableCookies Boolean"
"enableProtocolSwitchRewriting Boolean"
"enableSSLTracking Boolean"
"enableSecurityIntegration Boolean"
"enableUrlRewriting Boolean"
"maxWaitTime Integer"
"properties Property(TypedProperty)*"
"sessionDRSPersistence DRSSettings"
"sessionDatabasePersistence SessionDatabasePersistence"
"sessionPersistenceMode ENUM(DATABASE, DATA_REPLICATION, NONE)"
"tuningParams TuningParams"
```

4. Set up the attributes for session manager. The following example sets four top level attributes in the session manager. You can modify the example to set other attributes in the session manager including the nested attributes in Cookie, DRSSettings, SessionDataPersistence, and TuningParms object types. To list the attributes for those object types, use the **attributes** command of AdminConfig object.

   - Using Jacl:

     ```
     set attr0 [list enable true]
     set attr1 [list enableSecurityIntegration true]
     set attr2 [list maxWaitTime 30]
     set attr3 [list sessionPersistenceMode NONE]
     set attr4 [list enableCookies true]
     set attr5 [list invalidationTimeout 45]
     set tuningParmsDetailList [list $attr5]
     set tuningParamsList [list tuningParams $tuningParmsDetailList]
     set pwdList [list password 95ee608]
     set userList [list userId Administrator]
     set dsNameList [list datasourceJNDIName jdbc/session]
     set dbPersistenceList [list $dsNameList $userList $pwdList]
     set sessionDBPersistenceList [list $dbPersistenceList]
     set sessionDBPersistenceList [list sessionDatabasePersistence $dbPersistenceList]
     set kuki [list maximumAge 1000]
     set cookie [list $kuki]
     set cookieSettings [list defaultCookieSettings $cookie]
     set sessionManagerDetailList [list $attr0 $attr1 $attr2 $attr3 $attr4 $cookieSettings
       $tuningParamsList $sessionDBPersistenceList]
     set sessionMgr [list sessionManagement $sessionManagerDetailList]
     set id [$AdminConfig create ApplicationConfig $appDeploy [list $sessionMgr] configs]
     set targetMappings [lindex [$AdminConfig showAttribute $appDeploy targetMappings] 0]
     set attrs [list config $id]
     $AdminConfig modify $targetMappings [list $attrs]
     ```

     Example output using Jacl:

     ```
     sessionManagement {{enableSecurityIntegration true} {maxWaitTime 30} {sessionPersistenceMode NONE} {enabled true}}
     ```

   - Using Jython:

     ```
     attr0 = ['enable', 'true']
     attr1 = ['enableSecurityIntegration', 'true']
     attr2 = ['maxWaitTime', 30]
     attr3 = ['sessionPersistenceMode', 'NONE']
     attr4 = ['enableCookies', 'true']
     attr5 = ['invalidationTimeout', 45]
     tuningParmsDetailList = [attr5]
     tuningParamsList = ['tuningParams', tuningParmsDetailList]
     pwdList = ['password', '95ee608']
     userList = ['userId', 'Administrator']
     dsNameList = ['datasourceJNDIName', 'jdbc/session']
     dbPersistenceList = [dsNameList, userList, pwdList]
     sessionDBPersistenceList = [dbPersistenceList]
     sessionDBPersistenceList = ['sessionDatabasePersistence', dbPersistenceList]
     kuki = ['maximumAge', 1000]
     cookie = [kuki]
     cookieSettings = ['defaultCookieSettings', cookie]
     sessionManagerDetailList = [attr0, attr1, attr2, attr3, attr4, cookieSettings, tuningParamsList,
       sessionDBPersistenceList]
     sessionMgr = ['sessionManagement', sessionManagerDetailList]
     ```

```
id = AdminConfig.create('ApplicationConfig', appDeploy,[sessionMgr], 'configs')
targetMappings = AdminConfig.showAttribute(appDeploy, 'targetMappings')
targetMappings = targetMappings[1:len(targetMappings)-1]
print targetMappings
attrs = ['config', id]
AdminConfig.modify(targetMappings,[attrs])
```

Example output using Jython:

```
[sessionManagement, [[enableSecurityIntegration, true], [maxWaitTime, 30], [sessionPersistenceMode, NONE]]
```

5. Set up the attributes for Web module. For example:
   - Using Jacl:

     ```
     set nameAttr [list name myWebModuleConfig]
     set descAttr [list description "Web Module config post create"]
     set webAttrs [list $nameAttr $descAttr $sessionMgr]
     ```

     Example output:

     ```
     {name myWebModuleConfig} {description {Web Module config post create}}
     {sessionManagement {{enableSecurityIntegration true} {maxWaitTime 30}
     {sessionPersistenceMode NONE} {enabled true}}}
     ```
   - Using Jython:

     ```
     nameAttr = ['name', 'myWebModuleConfig']
     descAttr = ['description', "Web Module config post create"]
     webAttrs = [nameAttr, descAttr, sessionMgr]
     ```

     Example output:

     ```
     [[name, myWebModuleConfig], [description, "Web Module config post create"],
     [sessionManagement, [[enableSecurityIntegration, true], [maxWaitTime, 30],
     [sessionPersistenceMode, NONE], [enabled, true]]]]
     ```

   where:

set	is a Jacl command
nameAttr, descAttr, webAttrs	are variable names
$	is a Jacl operator for substituting a variable name with its value
name	is an attribute
myWebModuleConfig	is a value of the name attribute
description	is an attribute
Web Module config post create	is a value of the description attribute

6. Create the session manager for each Web module in the application. You can modify the following example to set other attributes of session manager in Web module configuration.
   - Using Jacl:

     ```
     foreach module $mod1 {
          if ([regexp WebModuleDeployment $module] == 1} {
             $AdminConfig create WebModuleConfig $module $webAttrs
        $AdminConfig save
          }
       }
     ```
   - Using Jython:

     ```
     arrayModules = mod1[1:len(mod1)-1].split(" ")
     for module in arrayModules:
      if module.find('WebModuleDeployment') != -1:
       AdminConfig.create('WebModuleConfig', module, webAttrs)
       Adminconfig.save()
     ```

   Example output:

   ```
   myWebModuleConfig(cells/mycell/applications/myApp.ear/deployments/myApp|deployment.xml#WebModuleConfiguration_1)
   ```

7. Save the configuration changes. See the “Saving configuration changes with the wsadmin tool” on page 232 article for more information.
8. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 article for more information.

Exporting applications using scripting

You can export your applications before you update installed applications or before you migrate to a different version of the WebSphere Application Server product.

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

Exporting applications enables you to back them up and preserve their binding information.

- Export an enterprise application to a location of your choice, for example:
  - Using Jacl:

    ```
    $AdminApp export app1 C:/mystuff/exported.ear
    ```
  - Using Jython:

    ```
    AdminApp.export('app1', 'C:/mystuff/exported.ear')
    ```

  where:

`$`	is a Jacl operator for substituting a variable name with its value
`AdminApp`	is an object allowing application objects management
`export`	is an AdminApp command
app1	is the name of the application that will be exported
`/mystuff/exported.ear`	is the name of the file where the exported application will be stored

- Export Data Definition Language (DDL) files in the enterprise bean module of an application to a destination directory, for example:
  - Using Jacl:

    ```
    $AdminApp exportDDL app1 C:/mystuff
    ```
  - Using Jython:

    ```
    AdminApp.exportDDL('app1', 'C:/mystuff')
    ```

  where:

`$`	is a Jacl operator for substituting a variable name with its value
`AdminApp`	is an object allowing application objects management
`exportDDL`	is an AdminApp command
app1	is the name of the application whose DDL files will be exported
/mystuff	is the name of the directory where the DDL files export from the application

Configuring a shared library using scripting

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

Perform the following steps to configure an application server to use a shared library.

1. Identify the server and assign it to the server variable. For example:
   - Using Jacl:

     ```
     set serv [$AdminConfig getid /Cell:mycell/Node:mynode/Server:server1/]
     ```

   - Using Jython:

     ```
     serv = AdminConfig.getid('/Cell:mycell/Node:mynode/Server:server1/')
     print serv
     ```

   where:

`set`	is a Jacl command
`serv`	is a variable name
`$`	is a Jacl operator for substituting a variable name with its value
`AdminConfig`	is an object representing the WebSphere Application Server configuration
`getid`	is an AdminConfig command
`Cell`	is an attribute
`mycell`	is the value of the attribute
`Node`	is an attribute
`mynode`	is the value of the attribute
`Server`	is an attribute
`server1`	is the value of the attribute

   Example output:

   ```
   server1(cells/mycell/nodes/mynode/servers/server1|server.xml#Server_1)
   ```

2. Create the shared library in the server. For example:
   - Using Jacl:

     ```
     $AdminConfig create Library $serv {{name mySharedLibrary}
      {classPath c:/mySharedLibraryClasspath}}
     ```

   - Using Jython:

     ```
     print AdminConfig.create('Library', serv, [['name', 'mySharedLibrary'],
      ['classPath',  'c:/mySharedLibraryClasspath']])
     ```

   where:

`$`	is a Jacl operator for substituting a variable name with its value
`AdminConfig`	is an object representing the WebSphere Application Server configuration
`create`	is an AdminConfig command
`Library`	is an attribute
`serv`	evaluates the ID of the server specified in step number 1
`name`	is an attribute
`mySharedLibrary`	is a value of the name attribute
`classPath`	is an attribute
/`mySharedLibraryClasspath`	is the value of the classpath attribute
`print`	is a Jython command

   Example output:

```
MysharedLibrary(cells/mycell/nodes/mynode/servers/server1|libraries.xml#Library_1)
```

3. Identify the application server from the server and assign it to the appServer variable. For example:
   - Using Jacl:

     ```
     set appServer [$AdminConfig list ApplicationServer $serv]
     ```

   - Using Jython:

     ```
     appServer = AdminConfig.list('ApplicationServer', serv)
     print appServer
     ```

   where:

`set`	is a Jacl command
`appServer`	is a variable name
`$`	is a Jacl operator for substituting a variable name with its value
`AdminConfig`	is an object representing the WebSphere Application Server configuration
`list`	is an AdminConfig command
`ApplicationServer`	is an attribute
`serv`	evaluates the ID of the server specified in step number 1
`print`	is a Jython command

   Example output:

   ```
   server1(cells/mycell/nodes/mynode/servers/server1|server.xml#ApplicationServer_1
   ```

4. Identify the class loader in the application server and assign it to the classLoader variable. For example:
   - To use the existing class loader associated with the server, the following commands use the first class loader:
     – Using Jacl:

       ```
       set classLoad [$AdminConfig showAttribute $appServer classloaders]
       set classLoader1 [lindex $classLoad 0]
       ```

     – Using Jython:

       ```
       classLoad = AdminConfig.showAttribute(appServer, 'classloaders')
       cleanClassLoaders = classLoad[1:len(classLoad)-1]
       classLoader1 = cleanClassLoaders.split(' ')[0]
       ```

     where:

`set`	is a Jacl command
`classLoad`, `classLoader1`	is a variable name
`$`	is a Jacl operator for substituting a variable name with its value
`AdminConfig`	is an object representing the WebSphere Application Server configuration
`showAttribute`	is an AdminConfig command
`appServer`	evaluates the ID of the application server specified in step number 3
`classloaders`	is an attribute
`print`	is a Jython command

   - To create a new class loader, do the following step:
     – Using Jacl:

```
set classLoader1 [$AdminConfig create Classloader $appServer {{mode PARENT_FIRST}}]
```

– Using Jython:

```
classLoader1 = AdminConfig.create('Classloader', appServer, [['mode',  'PARENT_FIRST']])
```

where:

set	is a Jacl command
classLoader1	is a variable name
$	is a Jacl operator for substituting a variable name with its value
AdminConfig	is an object representing the WebSphere Application Server configuration
create	is an AdminConfig command
Classloader	is an attribute
appServer	evaluates the ID of the application server specified in step number 3
mode	is an attribute
PARENT_FIRST	is the value of the attribute
print	is a Jython command

Example output:

```
(cells/mycell/nodes/mynode/servers/server1|server.xml#Classloader_1)
```

5. Associate the created shared library with the application server through the class loader. For example:
   - Using Jacl:

     ```
     $AdminConfig create LibraryRef $classLoader1 {{libraryName MyshareLibrary} {sharedClassloader true}}
     ```

   - Using Jython:

     ```
     print AdminConfig.create('LibraryRef', classLoader1, [['libraryName', 'MyshareLibrary'],
      ['sharedClassloader', 'true']])
     ```

   where:

$	is a Jacl operator for substituting a variable name with its value
AdminConfig	is an object representing the WebSphere Application Server configuration
create	is an AdminConfig command
LibraryRef	is an attribute
classLoader1	evaluates the ID of the class loader specified in step number 4
libraryName	is an attribute
MyshareLibrary	is the value of the attribute
sharedClassloader	is an attribute
true	is the value of the attribute
print	is a Jython command

Example output:

```
(cells/mycell/nodes/mynode/servers/server1|server.xml#LibraryRef_1)
```

6. Save the configuration changes. See the “Saving configuration changes with the wsadmin tool” on page 232 article for more information.

7. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 article for more information.

Configuring a shared library for an application using scripting

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

You can use the AdminApp object to set certain configurations in the application. This example uses the AdminConfig object to configure a shared library for an application.

1. Identify the shared library and assign it to the library variable. You can either use an existing shared library or create a new one, for example:
   - To create a new shared library, perform the following steps:
     a. Idenitfy the node and assign it to a variable, for example:
        – Using Jacl:

        ```
        set n1 [$AdminConfig getid /Cell:mycell/Node:mynode/]
        ```

        – Using Jython:

        ```
        n1 = AdminConfig.getid('/Cell:mycell/Node:mynode/')
        print n1
        ```

        where:

`set`	is a Jacl command
`n1`	is a variable name
`$`	is a Jacl operator for substituting a variable name with its value
`AdminConfig`	is an object representing the WebSphere Application Server configuration
`getid`	is an AdminConfig command
`Cell`	is the object type
`mycell`	is the name of the object that will be modified
`Node`	is the object type
`mynode`	is the name of the object that will be modified

        Example output:

        ```
        mynode(cells/mycell/nodes/mynode|node.xml#Node_1)
        ```

     b. Create the shared library in the node. The following example creates a new shared library in the node scope. You can modify it to use the cell or server scope.
        – Using Jacl:

        ```
        set library [$AdminConfig create Library $n1 {{name mySharedLibrary} {classPath c:/mySharedLibraryClasspath
        ```

        – Using Jython:

        ```
        library = AdminConfig.create('Library', n1, [['name', 'mySharedLibrary'], ['classPath', 'c:/mySharedLibrary
        print library
        ```

        where:

`set`	is a Jacl command
`library`	is a variable name
`$`	is a Jacl operator for substituting a variable name with its value
`AdminConfig`	is an object representing the WebSphere Application Server configuration
`create`	is an AdminConfig command

`Library`	is an AdminConfig object
`n1`	evaluates to the ID of host node specified in step number 1
`name`	is an attribute
mySharedLibrary	is the value of the name attribute
`classPath`	is an attribute
/mySharedLibraryClasspath	is the value of the classPath attribute

Example output:

```
MySharedLibrary(cells/mycell/nodes/mynode|libraries.xml#Library_1)
```

- To use an existing shared library, issue the following command:
  - Using Jacl:

    ```
    set library [$AdminConfig getid /Library:mySharedLibrary/]
    ```

  - Using Jython:

    ```
    library = AdminConfig.getid('/Library:mySharedLibrary/')
    print library
    ```

  where:

`set`	is a Jacl command
`library`	is a variable name
`$`	is a Jacl operator for substituting a variable name with its value
`AdminConfig`	is an object representing the WebSphere Application Server configuration
`getid`	is an AdminConfig command
`Library`	is an attribute
mySharedLibrary	is the value of the Library attribute

Example output:

```
MySharedLibrary(cells/mycell/nodes/mynode|libraries.xml#Library_1)
```

2. Identify the deployment configuration object for the application and assign it to the deployment variable. For example:
   - Using Jacl:

     ```
     set deployment [$AdminConfig getid /Deployment:myApp/]
     ```

   - Using Jython:

     ```
     deployment = AdminConfig.getid('/Deployment:myApp/')
     print deployment
     ```

   where:

`set`	is a Jacl command
`deployment`	is a variable name
`$`	is a Jacl operator for substituting a variable name with its value
`AdminConfig`	is an object representing the WebSphere Application Server configuration
`getid`	is an AdminConfig command
`Deployment`	is an attribute
myApp	is the value of the Deployment attribute
`print`	is a Jython command

Example output:

```
myApp(cells/mycell/applications/myApp.ear/deployments/myApp|deployment.xml#Deployment_1)
```

3. Retrieve the application deployment and assign it to the appDeploy variable. For example:
   - Using Jacl:

     ```
     set appDeploy [$AdminConfig showAttribute $deployment deployedObject]
     ```

   - Using Jython:

     ```
     appDeploy = AdminConfig.showAttribute(deployment, 'deployedObject')
     print appDeploy
     ```

   where:

set	is a Jacl command
appDeploy	is a variable name
$	is a Jacl operator for substituting a variable name with its value
AdminConfig	is an object representing the WebSphere Application Server configuration
showAttribute	is an AdminConfig command
deployment	evaluates the ID of the deployment configuration object specified in step number 2
deployedObject	is an attribute of modify objects
print	is a Jython command

Example output:

```
(cells/mycell/applications/myApp.ear/deployments/myApp|deployment.xml#ApplicationDeployment_1)
```

4. Identify the class loader in the application deployment and assign it to the classLoader variable. For example:
   - Using Jacl:

     ```
     set classLoad1 [$AdminConfig showAttribute $appDeploy classloader]
     ```

   - Using Jython:

     ```
     classLoad1 = AdminConfig.showAttribute(appDeploy, 'classloader')
     print classLoad1
     ```

   where:

set	is a Jacl command
classLoad1	is a variable name
$	is a Jacl operator for substituting a variable name with its value
AdminConfig	is an object representing the WebSphere Application Server configuration
showAttribute	is an AdminConfig command
appDeploy	evaluates the ID of the application deployment specified in step number 3
classLoader	is an attribute of modify objects
print	is a Jython command

Example output:

```
(cells/mycell/applications/myApp.ear/deployments/myApp|deployment.xml#Classloader_1)
```

5. Associate the shared library in the application through the class loader. For example:

- Using Jacl:

  ```
  $AdminConfig create LibraryRef $classLoad1 {{libraryName MyshareLibrary} {sharedClassloader true}}
  ```

- Using Jython:

  ```
  print AdminConfig.create('LibraryRef', classLoad1, [['libraryName', 'MyshareLibrary'],  ['sharedClassloader', 'true']
  ```

where:

`$`	is a Jacl operator for substituting a variable name with its value
`AdminConfig`	is an object representing the WebSphere Application Server configuration
`create`	is an AdminConfig command
`LibraryRef`	is an AdminConfig object
`classLoad1`	evaluates to the ID of class loader specified in step number 4
`libraryName`	is an attribute
MyshareLibrary	is the value of the libraryName attribute
`sharedClassloader`	is an attribute
true	is the value of the sharedClassloader attribute

Example output:

```
(cells/mycell/applications/myApp.ear/deployments/myApp|deployment.xml#LibraryRef_1)
```

6. Save the configuration changes. See the "Saving configuration changes with the wsadmin tool" on page 232 article for more information.
7. In a network deployment environment only, synchronize the node. See the "Synchronizing nodes with the wsadmin tool" on page 217 article for more information.

Setting background applications using scripting

Before starting this task, the wsadmin tool must be running. See the "Starting the wsadmin scripting client" on page 250 article for more information.

Perform the following steps to enable or disable a background application. Background applications specify whether the application must initialize fully before the server starts. The default setting is `false` and this indicates that server startup will not complete until the application starts. If you set the value to `true`, the application starts on a background thread and server startup continues without waiting for the application to start. The application may not ready for use when the application server starts.

1. Locate the application deployment object for the application. For example:
   - Using Jacl:

     ```
     set applicationDeployment [$AdminConfig getid /Deployment:adminconsole/ApplicationDeployment:/]
     ```

   - Using Jython:

     ```
     applicationDeployment = AdminConfig.getid('/Deployment:adminconsole/ApplicationDeployment:/')
     ```

   where:

`set`	is a Jacl command
`applicationDeployment`	is a variable name
`$`	is a Jacl operator for substituting a variable name with its value
`AdminConfig`	is an object representing the WebSphere Application Server configuration
`getid`	is an AdminConfig command

`Deployment`	is a type
`ApplicationDeployment`	is a type
`adminconsole`	is the name of the application

2. Enable the background application. For example:
   - Using Jacl:
     ```
     $AdminConfig modify $applicationDeployment "{backgroundApplication true}"
     ```
   - Using Jython:
     ```
     AdminConfig.modify(applicationDeployment, ['backgroundApplication', 'true'])
     ```

   where:

`$`	is a Jacl operator for substituting a variable name with its value
`AdminConfig`	is an object representing the WebSphere Application Server configuration
`modify`	is an AdminConfig command
`applicationDeployment`	is a variable name that was set in step 1
`backgroundApplication`	is an attribute
`true`	is the value of the backgroundApplication attribute

3. Save the configuration changes. See the “Saving configuration changes with the wsadmin tool” on page 232 article for more information.
4. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 article for more information.

Configuring servers with scripting

This topic contains the following tasks:

- “Creating a server using scripting”
- “Configuring the Java virtual machine using scripting” on page 284
- “Configuring enterprise bean containers using scripting” on page 284
- “Configuring a Performance Manager Infrastructure service using scripting” on page 288
- “Limiting the growth of Java virtual machine log files using scripting” on page 290
- “Configuring an ORB service using scripting” on page 292
- “Configuring for processes using scripting” on page 293
- “Configuring transaction properties for a server using scripting” on page 295
- “Setting port numbers kept in the serverindex.xml file using scripting” on page 296
- “Disabling components using scripting” on page 297
- “Disabling services using scripting” on page 298
- “Dynamic caching with scripting” on page 299

Creating a server using scripting

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

Creating an application server involves a configuration command. Perform the following steps to create a server:

1. Obtain the configuration ID of the object and assign it to the `node` variable, for example:

Using Jacl:

```
set node [$AdminConfig getid /Node:mynode/]
```

Using Jython:

```
node = AdminConfig.getid('/Node:mynode/')
```

2. Create the server using the node that you specified in the first step:

   Using Jacl:

```
$AdminConfig create Server $node {{name myserv} {outputStreamRedirect {{fileName myfile.out}}}}
```

   Using Jython:

```
AdminConfig.create('Server', node, [['name', 'myserv'], ['outputStreamRedirect', [['fileName', 'myfile.out']]]])
```

3. Save the configuration changes. See the “Saving configuration changes with the wsadmin tool” on page 232 article for more information.
4. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 article for more information.

Configuring the Java virtual machine using scripting

An example modifying the Java virtual machine (JVM) of a server to turn on debug follows:

- Identify the server and assign it to the server1 variable.

  Using Jacl:

```
set server1 [$AdminConfig getid /Cell:mycell/Node:mynode/Server:server1/]
```

  Using Jython:

```
server1 = AdminConfig.getid('/Cell:mycell/Node:mynode/Server:server1/')
print server1
```

  Example output:

```
server1(cells/mycell/nodes/mynode/servers/server1|server.xml#Server_1)
```

- Identify the JVM belonging to this server and assign it to the jvm variable.

  Using Jacl:

```
set jvm [$AdminConfig list JavaVirtualMachine $server1]
```

  Using Jython:

```
jvm = AdminConfig.list('JavaVirtualMachine', server1)
print jvm
```

  Example output:

```
(cells/mycell/nodes/mynode/servers/server1:server.xml#JavaVirtualMachine_1)
```

- Modify the JVM to turn on debug.

  Using Jacl:

```
$AdminConfig modify $jvm {{debugMode true} {debugArgs "-Djava.compiler=NONE -Xdebug -Xnoagent
  -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=7777"}}
```

  Using Jython:

```
AdminConfig.modify(jvm, [['debugMode', 'true'], ['debugArgs',  "-Djava.compiler=NONE -Xdebug -Xnoagent
  -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=7777"]])
```

- Save the changes with the following command:

  Using Jacl:

```
$AdminConfig save
```

  Using Jython:

```
AdminConfig.save()
```

Configuring enterprise bean containers using scripting

Before starting this task, the wsadmin tool must be running. See “Starting the wsadmin scripting client” on page 250 for more information.

Perform the following steps to configure an enterprise bean container:

1. Identify the application server and assign it to the serv1 variable. For example:
   - Using Jacl:

     ```
     set serv1 [$AdminConfig getid /Cell:mycell/Node:mynode/Server:server1/]
     ```

   - Using Jython:

     ```
     serv1 = AdminConfig.getid('/Cell:mycell/Node:mynode/Server:server1/')
     print serv1
     ```

   where:

set	is a Jacl command
serv1	is a variable name
$	is a Jacl operator for substituting a variable name with its value
AdminConfig	is an object representing the WebSphere Application Server configuration
getid	is an AdminConfig command
/Cell:*mycell*/Node:*mynode*/Server:*server1*/	is the hierarchical containment path of the configuration object
Cell	is the object type
mycell	is the optional name of the object
Node	is the object type
mynode	is the optional name of the object
Server	is the object type
server1	is the optional name of the object

   Example output:

   ```
   server1(cells/mycell/nodes/mynode/servers/server1|server.xml#Server_1)
   ```

2. Identify the EJB container belonging to the server and assign it to the ejbc1 variable. For example:
   - Using Jacl:

     ```
     set ejbc1 [$AdminConfig list EJBContainer $serv1]
     ```

   - Using Jython:

     ```
     ejbc1 = AdminConfig.list('EJBContainer', serv1)
     print ejbc1
     ```

   where:

set	is a Jacl command
ejbc1	is a variable name
$	is a Jacl operator for substituting a variable name with its value
AdminConfig	is an object representing the WebSphere Application Server configuration
list	is an AdminConfig command
EJBContainer	is the object type **Note:** The name of the object type that you input here is the one based on the XML configuration files and does not have to be the same name that the administrative console displays.

serv1	evaluates to the ID of the server specified in step number 1

Example output:

```
(cells/mycell/nodes/mynode/servers/server1|server.xml#EJBContainer_1)
```

3. View all the attributes of the enterprise bean container.
   - The following example command does not show nested attributes:
     - Using Jacl:

       ```
       $AdminConfig show $ejbc1
       ```

       Example output:

       ```
       {cacheSettings (cells/mycell/nodes/mynode/servers/server1|server.xml#EJBCache_1)}
       {components {}}
       {inactivePoolCleanupInterval 30000}
       {parentComponent (cells/mycell/nodes/mynode/servers/server1|server.xml#ApplicationServer_1)
       {passivationDirectory ${USER_INSTALL_ROOT}/temp}
       {properties {}}
       {services {(cells/mycell/nodes/mynode/servers/server1|server.xml#MessageListenerService_1)}
       {stateManagement (cells/mycell/nodes/mynode/servers/server1|server.xml#StateManageable_10)}
       ```

     - Using Jython:

       ```
       print AdminConfig.show(ejbc1)
       ```

       Example output:

       ```
       [cacheSettings (cells/mycell/nodes/myode/servers/server1|server.xml#EJBCache_1)]
       [components []]
       [inactivePoolCleanupInterval 30000]
       [parentComponent (cells/mycell/nodes/myode/servers/server1|server.xml#ApplicationServer_1)
       [passivationDirectory ${USER_INSTALL_ROOT}/temp]
       [properties []]
       [services [(cells/mycell/nodes/myode/servers/server1|server.xml#MessageListenerService_1)]
       [stateManagement (cells/mycell/nodes/mynode/servers/server1|server.xml#StateManageable_10)]
       ```

   where:

$	is a Jacl operator for substituting a variable name with its value
print	is a Jython command
AdminConfig	is an object representing the WebSphere Application Server configuration
showall	is an AdminConfig command
ejbc1	evaluates to the ID of the enterprise bean container specified in step number 2

   - The following command example includes nested attributes:
     - Using Jacl:

       ```
       $AdminConfig showall $ejbc1
       ```

       Example output:

       ```
       {cacheSettings {{cacheSize 2053}
         {cleanupInterval 3000}}}
       {components {}}
       {inactivePoolCleanupInterval 30000}
       {parentComponent (cells/mycell/nodes/mynode/servers/server1|server.xml#ApplicationServer_1)}
       {passivationDirectory ${USER_INSTALL_ROOT}/temp}
       {properties {}}
       {services {{{context (cells/mycell/nodes/mynode/servers/server1|server.xml#EJBContainer_1)}
         {listenerPorts {}}
         {properties {}}
         {threadPool {{inactivityTimeout 3500}
       ```

```
      {isGrowable false}
      {maximumSize 50}
      {minimumSize 10}}}}}}
  {stateManagement {{initialState START}
    {managedObject (cells/mycell/nodes/mynode/servers/server1|server.xml#EJBContainer_1)}}}
```

– Using Jython:

```
print AdminConfig.showall(ejbc1)
```

Example output:

```
[cacheSettings [[cacheSize 2053]
  [cleanupInterval 3000]]]
[components []]
[inactivePoolCleanupInterval 30000]
[parentComponent (cells/mycell/nodes/mynode/servers/server1|server.xml#ApplicationServer_1)]
[passivationDirectory ${USER_INSTALL_ROOT}/temp]
[properties []]
[services [[[context (cells/mycell/nodes/mynode/servers/server1|server.xml#EJBContainer_1)]
  [listenerPorts []]
  [properties []]
  [threadPool [[inactivityTimeout 3500]
    [isGrowable false]
    [maximumSize 50]
    [minimumSize 10]]]]]]
[stateManagement {{initialState START]
  [managedObject (cells/mycell/nodes/mynode/servers/server1|server.xml#EJBContainer_1)]]]
```

where:

$	is a Jacl operator for substituting a variable name with its value
print	is a Jython command
AdminConfig	is an object representing the WebSphere Application Server configuration
showall	is an AdminConfig command
ejbc1	evaluates to the ID of the enterprise bean container specified in step number 2

4. Modify the attributes.
   - The following example modifies the enterprise bean cache settings and it includes nested attributes:
     – Using Jacl:

```
$AdminConfig modify $ejbc1 {{cacheSettings {{cacheSize 2500} {cleanupInterval 3500}}}}
```

– Using Jython:

```
AdminConfig.modify(ejbc1, [['cacheSettings', [['cacheSize', 2500],  ['cleanupInterval', 3500]]]])
```

where:

$	is a Jacl operator for substituting a variable name with its value
AdminConfig	is an object representing the WebSphere Application Server configuration
modify	is an AdminConfig command
ejbc1	evaluates to the ID of the enterprise bean container specified in step number 2
cacheSettings	is an attribute of modify objects
cacheSize	is an attribute of modify objects
2500	is the value of the cacheSize attribute

`cleanupInterval`	is an attribute of modify objects
`3500`	is the value of the cleanupInterval attribute

- The following example modifies the cleanup interval attribute:
  - Using Jacl:

    ```
    $AdminConfig modify $ejbc1 {{inactivePoolCleanupInterval 15000}}
    ```

  - Using Jython:

    ```
    AdminConfig.modify(ejbc1, [['inactivePoolCleanupInterval', 15000]])
    ```

  where:

`$`	is a Jacl operator for substituting a variable name with its value
`AdminConfig`	is an object representing the WebSphere Application Server configuration
`modify`	is an AdminConfig command
`ejbc1`	evaluates to the ID of the enterprise bean container specified in step number 2
`inactivePoolCleanupInterval`	is an attribute of modify objects
`15000`	is the value of the inactivePoolCleanupInterval attribute

5. Save the changes. For example:
   - Using Jacl:

     ```
     $AdminConfig save
     ```

   - Using Jython:

     ```
     AdminConfig.save()
     ```

   where:

`$`	is a Jacl operator for substituting a variable name with its value
`AdminConfig`	is an object representing the WebSphere Application Server configuration
`save`	is an AdminConfig command

Configuring a Performance Manager Infrastructure service using scripting

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

Use the following steps to configure the Performance Manager Infrastructure (PMI) service for an application server:

1. Identify the application server and assign it to the s1 variable, for example:
   - Using Jacl:

     ```
     set s1 [$AdminConfig getid /Cell:mycell/Node:mynode/Server:server1/]
     ```

   - Using Jython:

     ```
     s1 = AdminConfig.getid('Cell:mycell/Node:mynode/Server:server1/')
     ```

   where:

`set`	is a Jacl command

s1	is a variable name
$	is a Jacl operator for substituting a variable name with its value
AdminConfig	is an object representing the WebSphere Application Server configuration
getid	is an AdminConfig command
Cell	is an attribute
mycell	is the value of the Cell attribute
Node	is an attribute
mynode	is the value of the Node attribute
Server	is an attribute
server1	is the value of the Server attribute

Example output:

```
server1(cells/mycell/nodes/mynode/servers/server1|server.xml#Server_1)
```

2. Identify the PMI service that belongs to the server and assign it to the pmi variable, for example:
   - Using Jacl:

     ```
     set pmi [$AdminConfig list PMIService $s1]
     ```

   - Using Jython:

     ```
     pmi = AdminConfig.list('PMIService', s1)
     print pmi
     ```

   where:

set	is a Jacl command
pmi	is a variable name
$	is a Jacl operator for substituting a variable name with its value
AdminConfig	is an object representing the WebSphere Application Server configuration
list	is an AdminConfig command
PMIService	is an AdminConfig object
s1	evaluates to the ID of the application server specified in step number 1

Example output:

```
(cells/mycell/nodes/mynode/servers/server1|server.xml#PMIService_1)
```

3. Modify the attributes, for example:
   - Using Jacl:

     ```
     $AdminConfig modify $pmi {{enable true}
     {initialSpecLevel beanModule=H:cacheModule=H:connectionPoolModule=
     H:j2cModule=H:jvmRuntimeModule=H:orbPerfModule=H:servletSessionsModule
     =H:systemModule=H:threadPoolModule=H:transactionModule=H:
     webAppModule=H:webServicesModule=H:wlmModule=H:wsgwModule=H}}
     ```

   - Using Jython:

     ```
     AdminConfig.modify(pmi, [['enable', 'true'],
     ['initialSpecLevel', 'beanModule=H:cacheModule=H:connectionPoolModule=
     H:j2cModule=H:jvmRuntimeModule=H:orbPerfModule=H:servletSessionsModule=
     H:systemModule=H:threadPoolModule=H:transactionModule=H:webAppModule=H:
     webServicesModule=H:wlmModule=H:wsgwModule=H']])
     ```

This example enables PMI service and sets the specification levels for all of components in the server. The following are the valid specification levels for the components:

N	represents **none**
L	represents **low**
M	represents **medium**
H	represents **high**
X	represents **maximum**

4. Save the configuration changes. See the “Saving configuration changes with the wsadmin tool” on page 232 article for more information.
5. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 article for more information.

Limiting the growth of Java virtual machine log files using scripting

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

Use the following example to configure the rotation policy settings for Java virtual machine (JVM) logs:

1. Identify the application server and assign it to the server1 variable, for example:
   - Using Jacl:
     ```
     set s1 [$AdminConfig getid /Cell:mycell/Node:mynode/Server:server1/]
     ```
   - Using Jython:
     ```
     s1 = AdminConfig.getid('/Cell:mycell/Node:mynode/Server:server1/')
     print s1
     ```

   where:

`set`	is a Jacl command
`s1`	is a variable name
`$`	is a Jacl operator for substituting a variable name with its value
`AdminConfig`	is an object representing the WebSphere Application Server configuration
`getid`	is an AdminConfig command
`Cell`	is the object type
`mycell`	is the name of the object that will be modified
`Node`	is the object type
`mynode`	is the name of the object that will be modified
`Server`	is the object type
`server1`	is the name of the object that will be modified
`print`	a Jython command

   Example output:
   ```
   server1(cells/mycell/nodes/mynode/servers/server1|server.xml#Server_1)
   ```
2. Identify the stream log and assign it to the log variable, for example:
   - The following example identifies the output stream log:
     – Using Jacl:
       ```
       set log [$AdminConfig showAttribute $s1 outputStreamRedirect]
       ```

- Using Jython:

  ```
  log = AdminConfig.showAttribute(s1, 'outputStreamRedirect')
  ```

- The following example identifies the error stream log:
  - Using Jacl:

    ```
    set log [$AdminConfig showAttribute $s1 errorStreamRedirect]
    ```

  - Using Jython:

    ```
    log = AdminConfig.showAttribute(s1, 'errorStreamRedirect')
    ```

Example output:

```
(cells/mycell/nodes/mynode/servers/server1|server.xml#StreamRedirect_2)
```

3. List the current values of the stream log, for example:
   - Using Jacl:

     ```
     $AdminConfig show $log
     ```

   - Using Jython:

     ```
     AdminConfig.show(log)
     ```

   Example output:

   ```
   {baseHour 24}
   {fileName ${SERVER_LOG_ROOT}/SystemOut.log}
   {formatWrites true}
   {maxNumberOfBackupFiles 1}
   {messageFormatKind BASIC}
   {rolloverPeriod 24}
   {rolloverSize 1}
   {rolloverType SIZE}
   {suppressStackTrace false}
   {suppressWrites false}
   ```

4. Modify the rotation policy for the stream log.
   - The following example sets the rotation log file size to two megabytes:
     - Using Jacl:

       ```
       $AdminConfig modify $log {{rolloverSize 2}}
       ```

     - Using Jython:

       ```
       AdminConfig.modify(log, ['rolloverSize', 2])
       ```

   - The following example sets the rotation policy to manage itself. It is based on the age of the file with the rollover algorithm loaded at midnight, and the log file rolling over every 12 hours:
     - Using Jacl:

       ```
       $AdminConfig modify $log {{rolloverType TIME} {rolloverPeriod 12} {baseHour 24}}
       ```

     - Using Jython:

       ```
       AdminConfig.modify(log, [['rolloverType', 'TIME'] ['rolloverPeriod', 12] ['baseHour', 24]])
       ```

   - The following example sets the log file to roll over based on both time and size:
     - Using Jacl:

       ```
       $AdminConfig modify $log {{rolloverType BOTH} {rolloverSize 2} {rolloverPeriod 12} {baseHour 24}}
       ```

     - Using Jython:

       ```
       AdminConfig.modify(log, [['rolloverType', 'BOTH'] ['rolloverSize', 2] ['rolloverPeriod', 12] ['baseHour', 24]])
       ```

5. Save the configuration changes. See the “Saving configuration changes with the wsadmin tool” on page 232 article for more information.
6. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 article for more information.

Configuring an ORB service using scripting

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

Perform the following steps to modify the Object Request Broker (ORB) service for an application server:

1. Identify the application server and assign it to the server variable:
   - Using Jacl:
     ```
     set s1 [$AdminConfig getid /Cell:mycell/Node:mynode/Server:server1/]
     ```
   - Using Jython:
     ```
     s1 = AdminConfig.getid('/Cell:mycell/Node:mynode/Server:server1/')
     print s1
     ```

   where:

`set`	is a Jacl command
`s1`	is a variable name
`$`	is a Jacl operator for substituting a variable name with its value
`AdminConfig`	is an object representing the WebSphere Application Server configuration
`getid`	is an AdminConfig command
`Cell`	is the object type
`mycell`	is the name of the object that will be modified
`Node`	is the object type
`mynode`	is the name of the object that will be modified
`Server`	is the object type
`server1`	is the name of the object that will be modified
`print`	a Jython command

   Example output:
   ```
   server1(cells/mycell/nodes/mynode/servers/server1|server.xml#Server_1)
   ```

2. Identify the ORB belonging to the server and assign it to the orb variable:
   - Using Jacl:
     ```
     set orb [$AdminConfig list ObjectRequestBroker $s1]
     ```
   - Using Jython:
     ```
     orb = AdminConfig.list('ObjectRequestBroker', s1)
     print orb
     ```

   where:

`set`	is a Jacl command
`orb`	is a variable name
`$`	is a Jacl operator for substituting a variable name with its value
`AdminConfig`	is an object representing the WebSphere Application Server configuration
`list`	is an AdminConfig command
`ObjectRequestBroker`	is an AdminConfig object
`s1`	evaluates to the ID of server specified in step number 1

print	a Jython command

Example output:

```
(cells/mycell/nodes/mynode/servers/server1|server.xml#ObjectRequestBroker_1)
```

3. Modify the attributes. The following example modifies the connection cache maximum and pass by value attributes. You can modify the example to change the value of other attributes.
   - Using Jacl:
     ```
     $AdminConfig modify $orb {{connectionCacheMaximum 252} {noLocalCopies true}}
     ```
   - Using Jython:
     ```
     AdminConfig.modify(orb, [['connectionCacheMaximum', 252], ['noLocalCopies',  'true']])
     ```

   where:

$	is a Jacl operator for substituting a variable name with its value
AdminConfig	is an object representing the WebSphere Application Server configuration
modify	is an AdminConfig command
orb	evaluates to the ID of ORB specified in step number 2
connectionCacheMaximum	is an attribute
252	is the value of the connectionCacheMaximum attribute
noLocalCopies	is an attribute
true	is the value of the noLocalCopies attribute

4. Save the configuration changes. See the “Saving configuration changes with the wsadmin tool” on page 232 article for more information.
5. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 article for more information.

Configuring for processes using scripting

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

Perform the following steps to configure a process:

1. Identify the server and assign it to the s1 variable. For example:
   - Using Jacl:
     ```
     set s1 [$AdminConfig getid /Cell:mycell/Node:mynode/Server:server1/]
     ```
   - Using Jython:
     ```
     s1 = AdminConfig.getid('/Cell:mycell/Node:mynode/Server:server1/')
     print s1
     ```

   where:

set	is a Jacl command
s1	is a variable name
$	is a Jacl operator for substituting a variable name with its value
AdminConfig	is an object representing the WebSphere Application Server configuration
getid	is an AdminConfig command

`Cell`	is the object type
`mycell`	is the name of the object that will be modified
`Node`	is the object type
`mynode`	is the name of the object that will be modified
`Server`	is the object type
`server1`	is the name of the object that will be modified
`print`	a Jython command

Example output:

```
server1(cells/mycell/nodes/mynode/servers/server1|server.xml#Server_1)
```

2. Identify the process definition belonging to this server and assign it to the processDef variable. For example:
   - Using Jacl:

     ```
     set processDef [$AdminConfig list JavaProcessDef $s1]
     set processDef [$AdminConfig showAttribute $s1 processDefinition]
     ```
   - Using Jython:

     ```
     processDef = AdminConfig.list('JavaProcessDef', s1)
     print processDef
     processDef = AdminConfig.showAttribute(s1, 'processDefinition')
     ```

   Example output:

   ```
   (cells/mycell/nodes/mynode/servers/server1|server.xml#JavaProcessDef_1)
   ```
3. Change the attributes.
   - On distributed systems, the following example changes the working directory.
     - Using Jacl:

       ```
       $AdminConfig modify $processDef {{workingDirectory c:/temp/user1}}
       ```
     - Using Jython:

       ```
       AdminConfig.modify(processDef, [['workingDirectory', 'c:/temp/user1']])
       ```
   - The following example modifies the stderr file name:
     - Using Jacl:

       ```
       set errFile [list stderrFilename \${LOG_ROOT}/server1/new_stderr.log]
       set attr [list $errFile]
       $AdminConfig modify $processDef [subst {{ioRedirect {$attr}}}]
       ```
     - Using Jython:

       ```
       errFile = ['stderrFilename', '\${LOG_ROOT}/server1/new_stderr.log']
       attr = [errFile]
       AdminConfig.modify(processDef, [['ioRedirect', [attr]]])
       ```
   - The following example modifies the process priority:
     - Using Jacl:

       ```
       $AdminConfig modify $processDef {{execution {{processPriority 15}}}}
       ```
     - Using Jython:

       ```
       AdminConfig.modify(processDef, [['execution', [['processPriority', 15]]]])
       ```
   - The following example changes the maximum startup attempts. You can modify this example to change other attributes in the process definition object.
     - Using Jacl:

       ```
       $AdminConfig modify $processDef {{monitoringPolicy {{maximumStartupAttempts 1}}}}
       ```
     - Using Jython:

       ```
       AdminConfig.modify(processDef, [['monitoringPolicy', [['maximumStartupAttempts',  1]]]])
       ```
4. Save the configuration changes. See the “Saving configuration changes with the wsadmin tool” on page 232 article for more information.
5. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 article for more information.

Configuring transaction properties for a server using scripting

Before starting this task, the wsadmin tool must be running. See the "Starting the wsadmin scripting client" on page 250 article for more information.

Perform the following steps to configure the runtime transaction properties for an application server.

1. Identify the transaction service MBean for the application server. The following command returns the transaction service MBean for *server1*.

   - Using Jacl:

     ```
     set ts [$AdminControl completeObjectName cell=mycell,node=mynode,process=server1,type=TransactionService,*]
     ```

   - Using Jython:

     ```
     ts = AdminControl.completeObjectName('cell=mycell,node=mynode,process=server1,type=TransactionService,*')
     print ts
     ```

   where:

set	is a Jacl command
ts	is a variable name
$	is a Jacl operator for substituting a variable name with its value
AdminControl	is an object that enables the manipulation of MBeans running in a WebSphere server process
completeObjectName	is an AdminControl command

cell=*mycell*,node=*mynode*,process=*server1*,
type=TransactionService
	is a fragment of the object name whose complete name is returned by this command. It is used to find the matching object name which is, in this case, the transaction object MBean for the node *mynode*, where *mynode* is the name of the node that you use to synchronize configuration changes. For example: type=TransactionService, process=server1. It can be any valid combination of domain and key properties. For example, type, name, cell, node, process, etc.

   Example output:

   ```
   WebSphere:cell=mycell,name=TransactionService,mbeanIdentifier=TransactionService,type=TransactionService,
    node=mynode,process=server1
   ```

2. Modify the attributes.

   - Using Jacl:

     ```
     $AdminControl invoke $ts {{transactionLogDirectory c:/WebSphere/AppServer/tranlog/server1}
      {clientInactivityTimeout 30} {totalTranLifetimeTimeout 180}}
     ```

   - Using Jython:

     ```
     AdminControl.invoke(ts, [['transactionLogDirectory',  'c:/WebSphere/AppServer/tranlog/server1'],
      ['clientInactivityTimeout', 30],  ['totalTranLifetimeTimeout', 180]])
     ```

   where:

$	is a Jacl operator for substituting a variable name with its value
AdminControl	is an object that enables the manipulation of MBeans running in a WebSphere server process
invoke	is an AdminControl command
ts	evaluates to the ID of the transaction service specified in step number 1
transactionLogDirectory	is an attribute

c:/WebSphere/AppServer/tranlog/server1	is the value of the transactionLogDirectory attribute
clientInactivityTimeout	is an attribute
30	is the value of the clientInactivityTimeout attribute specified in seconds. A value of 0 means that there is no timeout limit.
totalTranLifetimeTimeout	is an attribute
180	is the value of the totalTranLifetimeTimeout attribute specified in milliseconds. A value of 0 means that there is no timeout limit.

Setting port numbers kept in the serverindex.xml file using scripting

Before starting this task, the wsadmin tool must be running. See the "Starting the wsadmin scripting client" on page 250 article for more information.

This topic provides reference information about modifying port numbers in the `serverindex.xml` file. The end points of the `serverindex.xml` file are part of different objects in the configuration.

Use the following attributes to modify the end point information of the end point attributes for a process:

- **`BOOTSTRAP_ADDRESS`** `of server1 process`

  An attribute of the NameServer object that exists inside the server. It is used by the naming client to specify the naming server to look up the initial context. To modify its end point, obtain the ID of the NameServer object and issue a **modify** command, for example:

  – Using Jacl:

  ```
  set s [$AdminConfig getid /Cell:mycell/Node:mynode/Server:server1/]
  set ns [$AdminConfig list NameServer $s]
  $AdminConfig modify $ns {{BOOTSTRAP_ADDRESS {{port 2810} {host myhost}}}}
  ```

  – Using Jython:

  ```
  s = AdminConfig.getid('/Cell:mycell/Node:mynode/Server:server1/')
  ns = AdminConfig.list('NameServer', s)
  AdminConfig.modify(ns, [['BOOTSTRAP_ADDRESS', [['host', 'myhost'],  ['port', 2810]]]])
  ```

- **`SOAP_CONNECTOR-ADDRESS`** `of server1 process`

  An attribute of the SOAPConnector object that exists inside the server. It is the port that is used by HTTP transport for incoming SOAP requests. To modify its end point, obtain the ID of the SOAPConnector object and issue a modify command, for example:

  – Using Jacl:

  ```
  set s [$AdminConfig getid /Cell:mycell/Node:mynode/Server:server1/]
  set soap [$AdminConfig list SOAPConnector $s]
  $AdminConfig modify $soap {{SOAP_CONNECTOR_ADDRESS {{host myhost} {port 8881}}}}
  ```

  – Using Jython:

  ```
  s = AdminConfig.getid('/Cell:mycell/Node:mynode/Server:server1/')
  soap = AdminConfig.list('SOAPConnector', s)
  AdminConfig.modify(soap, [['SOAP_CONNECTOR_ADDRESS', [['host', 'myhost'],  ['port', 8881]]]])
  ```

- **`DRS_CLIENT_ADDRESS`** `of server1 process`

  An attribute of the SystemMessageServer object that exists inside the server. It is the port used to configure the Data Replication Service (DRS) which is a JMS-based message broker system for dynamic caching. The DRS_CLIENT_ADDRESS attribute is not available if a replication domain and a replicator entry have not been added to the server.

  To modify the end point of the DRS_CLIENT_ADDRESS attribute, obtain the ID of the SystemMessageServer object and issue a **modify** command, for example:

  – Using Jacl:

  ```
  set s [$AdminConfig getid /Cell:mycell/Node:mynode/Server:server1/]
  set sms [$AdminConfig list SystemMessageServer $s]
  $AdminConfig modify $sms {{DRS_CLIENT_ADDRESS {{host myhost} {port 7874}}}}
  ```

  - Using Jython:

    ```
    s = AdminConfig.getid('/Cell:mycell/Node:mynode/Server:server1/')
    sms = AdminConfig.list('SystemMessageServer', s)
    AdminConfig.modify(sms, [['DRS_CLIENT_ADDRESS', [['host', 'myhost'], ['port', 7874]]]])
    ```

- `JMSSERVER_QUEUED_ADDRESS and JMSSERVER_DIRECT_ADDRESS of server1 process`

  An attribute of the JMSServer object that exists inside the server. These are ports used to configure the WebSphere Application Server JMS provider topic connection factory settings. To modify its end point, obtain the ID of the JMSServer object and issue a **modify** command, for example:

  - Using Jacl:

    ```
    set s [$AdminConfig getid /Cell:mycell/Node:mynode/Server:server1/]
    set jmss [$AdminConfig list JMSServer $s]
    $AdminConfig modify $jmss {{JMSSERVER_QUEUED_ADDRESS {{host myhost} {port 5560}}}}
    $AdminConfig modify $jmss {{JMSSERVER_DIRECT_ADDRESS {{host myhost} {port 5561}}}}
    ```

  - Using Jython:

    ```
    s = AdminConfig.getid('/Cell:mycell/Node:mynode/Server:server1/')
    jmss = AdminConfig.list('JMSServer', s)
    AdminConfig.modify(jmss, [['JMSSERVER_QUEUED_ADDRESS', [['host', 'myhost'], ['port', 5560]]]])
    AdminConfig.modify(jmss, [['JMSSERVER_DIRECT_ADDRESS', [['host', 'myhost'], ['port', 5561]]]])
    ```

- `NODE_DISCOVERY_ADDRESS of nodeagent process`

  An attribute of the NodeAgent object that exists inside the server. It is the port used to receive the incoming process discovery messages inside a node agent process. To modify its end point, obtain the ID of the NodeAgent object and issue a **modify** command, for example:

  - Using Jacl:

    ```
    set nodeAgentServer [$AdminConfig getid /Cell:mycell/Node:mynode/Server:nodeagent/]
    set nodeAgent [$AdminConfig list NodeAgent $nodeAgentServer]
    $AdminConfig modify $nodeAgent {{NODE_DISCOVERY_ADDRESS {{host myhost} {port 7272}}}}
    ```

  - Using Jython:

    ```
    nodeAgentServer = AdminConfig.getid('/Cell:mycell/Node:mynode/Server:nodeagent/')
    nodeAgent = AdminConfig.list('NodeAgent', nodeAgentServer)
    AdminConfig.modify(nodeAgent, [['NODE_DISCOVERY_ADDRESS', [['host', 'myhost'], ['port', 7272]]]])
    ```

- `CELL_DISCOVERY_ADDRESS of dmgr process`

  An attribute of the deploymentManager object that exists inside the server. It is the port used to receive the incoming process discovery messages inside a deployment manager process. To modify its end point, obtain the ID of the deploymentManager object and issue a **modify** command, for example:

  - Using Jacl:

    ```
    set netmgr [$AdminConfig getid /Cell:mycell/Node:managernode/Server:dmgr/]
    set deploymentManager [$AdminConfig list CellManager $netmgr]
    $AdminConfig modify $deploymentManager {{CELL_MULTICAST_DISCOVERY_ADDRESS {{host myhost} {port 7272}}}}
    $AdminConfig modify $deploymentManager {{CELL_DISCOVERY_ADDRESS {{host myhost} {port 7278}}}}
    ```

  - Using Jython:

    ```
    netmgr = AdminConfig.getid('/Cell:mycell/Node:managernode/Server:dmgr/')
    deploymentManager = AdminConfig.list('CellManager', netmgr)
    AdminConfig.modify(deploymentManager, [['CELL_MULTICAST_DISCOVERY_ADDRESS', [['host', 'myhost'],  ['port', 7272]]]]
    AdminConfig.modify(deploymentManager, [['CELL_DISCOVERY_ADDRESS', [['host', 'myhost'], ['port', 7278]]]])
    ```

Save the configuration changes. See the “Saving configuration changes with the wsadmin tool” on page 232 article for more information.

In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 article for more information.

Disabling components using scripting

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

Perform the following steps to disable the name server component of a configured server. You can modify this example to disable a different component.

1. Identify the server component and assign it to the nameServer variable.
   - Using Jacl:
     ```
     set nameServer [$AdminConfig list NameServer $server]
     ```
   - Using Jython:
     ```
     nameServer = AdminConfig.list('NameServer', server)
     print nameServer
     ```

   Example output:
   ```
   (cells/mycell/nodes/mynode/servers/server1|server.xml#NameServer_1)
   ```
2. List the components belonging to the server and assign them to the components variable.
   - Using Jacl:
     ```
     set components [$AdminConfig list Component $server]
     ```
   - Using Jython:
     ```
     components = AdminConfig.list('Component', server)
     print components
     ```

   The components variable contains a list of components.

   Example output:
   ```
   (cells/mycell/nodes/mynode/servers/server1|server.xml#ApplicationServer_1)
   (cells/mycell/nodes/mynode/servers/server1|server.xml#EJBContainer_1)
   (cells/mycell/nodes/mynode/servers/server1|server.xml#NameServer_1)
   (cells/mycell/nodes/mynode/servers/server1|server.xml#WebContainer_1)
   ```
3. Identify the name server component and assign it to the nameServer variable.

   Since the name server component is the third element in the list, retrieve this element by using index 2.
   - Using Jacl:
     ```
     set nameServer [lindex $components 2]
     ```
   - Using Jython:
     ```
     # get line separator
     import  java
     lineSeparator = java.lang.System.getProperty('line.separator')
     arrayComponents = components.split(lineSeparator)
     nameServer = arrayComponents[2]
     print nameServer
     ```

   Example output:
   ```
   (cells/mycell/nodes/mynode/servers/server1|server.xml#NameServer_1)
   ```
4. Disable the name server component by changing the nested initialState attribute belonging to the stateManagement attribute. For example:
   - Using Jacl:
     ```
     $AdminConfig modify $nameServer {{stateManagement {{initialState STOP}}}}
     ```
   - Using Jython:
     ```
     AdminConfig.modify(nameServer, [['stateManagement', [['initialState', 'STOP']]]])
     ```
5. Save the configuration changes. See the "Saving configuration changes with the wsadmin tool" on page 232 article for more information.
6. In a network deployment environment only, synchronize the node. See the "Synchronizing nodes with the wsadmin tool" on page 217 article for more information.

Disabling services using scripting

Before starting this task, the wsadmin tool must be running. See the "Starting the wsadmin scripting client" on page 250 article for more information.

Perform the following steps to disable the trace service of a configured server. You can modify this example to disable a different service.

1. Identify the server and assign it to the server variable. For example:
   - 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. List all the services belonging to the server and assign them to the services variable. The following example returns a list of services:
   - Using Jacl:
     ```
     set services [$AdminConfig list Service $server]
     ```
   - Using Jython:
     ```
     services = AdminConfig.list('Service', server)
     print services
     ```

   Example output:
   ```
   (cells/mycell/nodes/mynode/servers/server1|server.xml#AdminService_1)
   (cells/mycell/nodes/mynode/servers/server1|server.xml#DynamicCache_1)
   (cells/mycell/nodes/mynode/servers/server1|server.xml#MessageListenerService_1)
   (cells/mycell/nodes/mynode/servers/server1|server.xml#ObjectRequestBroker_1)
   (cells/mycell/nodes/mynode/servers/server1|server.xml#PMIService_1)
   (cells/mycell/nodes/mynode/servers/server1|server.xml#RASLoggingService_1)
   (cells/mycell/nodes/mynode/servers/server1|server.xml#SessionManager_1)
   (cells/mycell/nodes/mynode/servers/server1|server.xml#TraceService_1)
   (cells/mycell/nodes/mynode/servers/server1|server.xml#TransactionService_1)
   ```
3. Identify the trace service and assign it to the traceService variable.

   Since trace service is the 7th element in the list, retrieve this element by using index 6.
   - Using Jacl:
     ```
     set traceService [$AdminConfig list TraceService $server]
     ```
   - Using Jython:
     ```
     traceService = AdminConfig.list('TraceService', server)
     print traceService
     ```

   Example output:
   ```
   (cells/mycell/nodes/mynode/servers/server1|server.xml#TraceService_1)
   ```
4. Disable the trace service by modifying the enable attribute. For example:
   - Using Jacl:
     ```
     $AdminConfig modify $traceService {{enable false}}
     ```
   - Using Jython:
     ```
     AdminConfig.modify(traceService, [['enable', 'false']])
     ```
5. Save the configuration changes. See the “Saving configuration changes with the wsadmin tool” on page 232 article for more information.
6. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 article for more information.

Dynamic caching with scripting

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

To see a list of parameters associated with dynamic caching, use the **attributes** command. For example:

```
$AdminConfig attributes DynamicCache
```

Perform the following steps to enable servlet caching:

1. Locate the server object. The following example selects the first server found:

   Using Jacl:

   ```
   set s1 [$AdminConfig getid /Server:server1/]
   ```

   Using Jython:

   ```
   s1 = AdminConfig.getid('/Server:server1/')
   ```

2. List the web containers and assign them to the `wc` variable, for example:

   Using Jacl:

   ```
   set wc [$AdminConfig list WebContainer $s1]
   ```

   Using Jython:

   ```
   wc = AdminConfig.list('WebContainer', s1)
   ```

3. Set the enableServletCaching attribute to `true` and assign it to the `serEnable` variable, for example:

   Using Jacl:

   ```
   set serEnable "{enableServletCaching true}"
   ```

   Using Jython:

   ```
   serEnable = [['enableServletCaching', 'true']]
   ```

4. Enable caching, for example:

   Using Jacl:

   ```
   $AdminConfig modify $wc $serEnable
   ```

   Using Jython:

   ```
   AdminConfig.modify(wc, serEnable)
   ```

Configuring connections to Webservers with scripting

This topic contains the following tasks:

- “Regenerating the node plug-in configuration using scripting”
- “Creating new virtual hosts using templates with scripting” on page 301

Regenerating the node plug-in configuration using scripting

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

Perform the following steps to regenerate the node plug-in configuration:

1. Identify the plug-in and assign it to the `generator` variable, for example:

   Using Jacl:

   ```
   set generator [$AdminControl completeObjectName type=PluginCfgGenerator,node=mynode,*]
   ```

   Using Jython:

   ```
   generator = AdminControl.completeObjectName('type=PluginCfgGenerator,node=mynode,*')
   ```

2. Regenerate the node plug-in:

   Using Jacl:

   ```
   $AdminControl invoke $generator generate "c:/WebSphere/DeloymentManager/config mycell mynode null plugin-cfg.xml"
   ```

   Using Jython:

   ```
   AdminControl.invoke(generator, 'generate', "c:/WebSphere/DeloymentManager/config mycell mynode null plugin-cfg.xml")
   ```

Creating new virtual hosts using templates with scripting

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

Some configuration object types have templates that you can use when you create a virtual host. You can create a new virtual host using a preexisting template or by creating a new custom template. Perform the following steps to create a new virtual host using a template:

1. If you want to create a new custom template, perform the following steps:
   a. Copy and paste the following file into a new file, *myvirtualhostname.xml*:
      ```
      <WAS-ROOT>\config\templates\default\virtualhosts.xml
      ```
   b. Edit and customize the new *myvirtualhostname.xml* file.
   c. Place the new file in the following directory:
      ```
      <WAS-ROOT>\config\templates\custom\
      ```

   If you want the new custom template to appear with the list of templates, restart the deployment manager on a network deployment edition, or use the AdminConfig object **reset** command. For example:
   - Using Jacl:
     ```
     $AdminConfig reset
     ```
   - Using Jython:
     ```
     AdminConfig.reset()
     ```

   The administrative console does not support the use of custom templates. The new template that you create will not be visible in the administrative console panels.
2. Use the AdminConfig object **listTemplates** command to list available templates, for example:
   - Using Jacl:
     ```
     $AdminConfig listTemplates VirtualHost
     ```
   - Using Jython:
     ```
     print AdminConfig.listTemplates('VirtualHost')
     ```

   Example output:
   ```
   default_host(templates/default:virtualhosts.xml#VirtualHost_1)
   my_host(templates/custom:virtualhostname.xml#VirtualHost_1)
   ```
3. Create a new virtual host. For example:
   - Using Jacl:
     ```
     set cell [$AdminConfig getid /Cell:NetworkDeploymentCell/]
     set vtempl [$AdminConfig listTemplates VirtualHost my_host]
     $AdminConfig createUsingTemplate VirtualHost $cell {{name newVirHost}} $vtempl
     ```
   - Using Jython:
     ```
     cell = AdminConfig.getid('/Cell:NetworkDeploymentCell/')
     vtempl = AdminConfig.listTemplates('VirtualHost', 'my_host')
     AdminConfig.createUsingTemplate('VirtualHost', cell, [['name', 'newVirHost']], vtempl)
     ```
4. Save the configuration changes. See the “Saving configuration changes with the wsadmin tool” on page 232 article for more information.
5. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 article for more information.

Managing servers with scripting

This topic contains the following tasks:

- “Stopping a node using scripting” on page 302
- “Starting servers using scripting” on page 302

- "Stopping servers using scripting" on page 303
- "Querying server state using scripting" on page 304
- "Listing running applications on running servers using scripting" on page 304
- "Starting listener ports using scripting" on page 306
- "Managing generic servers using scripting" on page 307
- "Setting development mode for server objects using scripting" on page 308
- "Disabling parallel startup using scripting" on page 308
- "Removing multicast endpoints using scripting" on page 309
- "Obtaining server version information with scripting" on page 309

Stopping a node using scripting

Before starting this task, the wsadmin tool must be running. See the "Starting the wsadmin scripting client" on page 250 article for more information.

Stopping the node agent on a remote machine process is an asynchronous action where the stop is initiated, and then control returns to the command line. Perform the following task to stop a node:

1. Identify the node that you want to stop and assign it to a variable:

   Using Jacl:

   ```
   set na [$AdminControl queryNames type=NodeAgent,node=mynode,*]
   ```

   Using Jython:

   ```
   na = AdminControl.queryNames('type=NodeAgent,node=mynode,*')
   ```

2. Stop the node:

   Using Jacl:

   ```
   $AdminControl invoke $na stopNode
   ```

   Using Jython:

   ```
   AdminControl.invoke(na, 'stopNode')
   ```

Starting servers using scripting

Before starting this task, the wsadmin tool must be running. See the "Starting the wsadmin scripting client" on page 250 article for more information.

Use the **startServer** command to start the server. This command has several syntax options. For example:

- To start a server on a WebSphere Application Server single server edition, choose one of the following options:
  - The following examples specify the server name only:

    Using Jacl:

    ```
    $AdminControl startServer serverName
    ```

    Using Jython:

    ```
    AdminControl.startServer('serverName')
    ```

  - The following example starts an application server with the node specified:
    - Using Jacl:

      ```
      $AdminControl startServer server1 mynode
      ```

    - Using Jython:

      ```
      print AdminControl.startServer('server1', 'mynode')
      ```

    Example output:

```
WASX7319I: The serverStartupSyncEnabled attribute is set to false.  A start
will be attempted for server "server1" but the configuration information for
node "mynode" may not be current.
WASX7262I: Start completed for server "server1" on node "mynode"
```

- The following example specify the server name and wait time:
  - Using Jacl:

    ```
    $AdminControl startServer serverName 10
    ```

  - Using Jython:

    ```
    AdminControl.startServer('serverName', 10)
    ```

  where *10* is the number of milliseconds that the process should wait before starting the server.

- To start a server on a WebSphere Application Server network deployment edition, choose one of the following options:
  - The following example specifies the server name and the node name:
    - Using Jacl:

      ```
      $AdminControl startServer serverName nodeName
      ```

    - Using Jython:

      ```
      AdminControl.startServer('serverName', 'nodeName')
      ```

  - The following example specifies the server name, the node name, and the wait time:
    - Using Jacl:

      ```
      $AdminControl startServer serverName nodeName 10
      ```

    - Using Jython:

      ```
      AdminControl.startServer('serverName', 'nodeName', 10)
      ```

    where *10* is the number of milliseconds that the process should wait before starting the server.

Stopping servers using scripting

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

Use the **stopServer** command to stop the server. This command has several syntax options. For example:

- To stop a server on a WebSphere Application Server single server edition, choose one of the following options:
  - The following examples specify the server name only:

    Using Jacl:

    ```
    $AdminControl stopServer serverName
    ```

    Using Jython:

    ```
    AdminControl.stopServer('serverName')
    ```

  - The following examples stop an application server with the node specified:
    - Using Jacl:

      ```
      $AdminControl stopServer serverName mynode
      ```

    - Using Jython:

      ```
      print AdminControl.stopServer('serverName', 'mynode')
      ```

    Example output:

    ```
    WASX7337I: Invoked stop for server "serverName" Waiting for stop completion.
    WASX7264I: Stop completed for server "serverName" on node "mynode"
    ```

  - The following examples specify the server name and immediate:
    - Using Jacl:

      ```
      $AdminControl stopServer serverName immediate
      ```

    - Using Jython:

```
AdminControl.stopServer('serverName', immediate)
```

- To stop a server on a WebSphere Application Server network deployment edition, choose one of the following options:
  - The following example specifies the server name and the node name:
    - Using Jacl:
      ```
      $AdminControl stopServer serverName nodeName
      ```
    - Using Jython:
      ```
      AdminControl.stopServer('serverName', 'nodeName')
      ```
  - The following example specifies the server name, the node name, and immediate:
    - Using Jacl:
      ```
      $AdminControl stopServer serverName nodeName immediate
      ```
    - Using Jython:
      ```
      AdminControl.stopServer('serverName', 'nodeName', immediate)
      ```

Querying server state using scripting

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

Perform the following steps to query the server state:

1. Identify the server and assign it to the server variable. The following example returns the server MBean that matches the partial object name string:
   - Using Jacl:
     ```
     set server [$AdminControl completeObjectName cell=mycell,node=mynode,name=server1,type=Server,*]
     ```
   - Using Jython:
     ```
     server = AdminControl.completObjectName('cell=mycell,node=mynode,name=server1,type=Server,*')
     print server
     ```

   Example output:
   ```
   WebSphere:cell=mycell,name=server1,mbeanIdentifier=server.xml#Server_1,type=Server,node=mynode,
    process=server1,processType=ManagedProcess
   ```
2. Query for the state attribute. For example:
   - Using Jacl:
     ```
     $AdminControl getAttribute $server state
     ```
   - Using Jython:
     ```
     print AdminControl.getAttribute(server, 'state')
     ```

   The **getAttribute** command returns the value of a single attribute.

   Example output:
   ```
   STARTED
   ```

Listing running applications on running servers using scripting

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

Use the following example to list all the running applications on all the running servers on each node of each cell:

- Using Jacl:
  ```
  *Provide this example as a Jacl script file and run it with  the "-f" option:
  1  #----------------------------------------------------------------
  2  # lines 4 and 5 find all the cell and process them one at a time
  3  #----------------------------------------------------------------
  ```

```
4  set cells [$AdminConfig list Cell]
5  foreach cell $cells {
6     #-----------------------------------------------------------------------
7     # lines 10 and 11 find all the nodes belonging to the cell and
8     # process them at a time
9     #-----------------------------------------------------------------------
10    set nodes [$AdminConfig list Node $cell]
11    foreach node $nodes {
12       #--------------------------------------------------------------
13       # lines 16-20 find all the running servers belonging to the cell
14       # and node, and process them one at a time
15       #--------------------------------------------------------------
16       set cname [$AdminConfig showAttribute $cell name]
17       set nname [$AdminConfig showAttribute $node name]
18       set servs [$AdminControl queryNames type=Server,cell=$cname,node=$nname,*]
19       puts "Number of running servers on node $nname: [llength $servs]"
20       foreach server $servs {
21           #---------------------------------------------------------
22           # lines 25-31 get some attributes from the server to display;
23           # invoke an operation on the server JVM to display a property.
24           #---------------------------------------------------------
25           set sname [$AdminControl getAttribute $server name]
26           set ptype [$AdminControl getAttribute $server processType]
27           set pid   [$AdminControl getAttribute $server pid]
28           set state [$AdminControl getAttribute $server state]
29           set jvm [$AdminControl queryNames type=JVM,cell=$cname,node=$nname,process=$sname,*]
30           set osname [$AdminControl invoke $jvm getProperty os.name]
31           puts "  $sname ($ptype) has pid $pid; state: $state; on $osname"
32
j3           #---------------------------------------------------------
34           # line 37-42 find the applications running on this server and
35           # display the application name.
35           #---------------------------------------------------------
37           set apps [$AdminControl queryNames type=Application,cell=$cname,node=$nname,process=$sname,*]
38           puts "  Number of applications running on $sname: [llength $apps]"
39           foreach app $apps {
40              set aname [$AdminControl getAttribute $app name]
41              puts "    $aname"
42           }
43           puts "-----------------------------------------------------"
44           puts ""
45
46        }
47     }
48  }
```

- Using Jython:

```
* Provide this example as a Jython script file and run it with the "-f" option:

1   #----------------------------------------------------------------
2   # lines 7 and 8 find all the cell and process them one at a time
3   #----------------------------------------------------------------
4   # get line separator
5   import  java.lang.System  as  sys
6   lineSeparator = sys.getProperty('line.separator')
7   cells = AdminConfig.list('Cell').split(lineSeparator)
8   for cell in cells:
9        #-----------------------------------------------------------------------
10      # lines 13 and 14 find all the nodes belonging to the cell and
11      # process them at a time
12      #-----------------------------------------------------------------------
13      nodes = AdminConfig.list('Node', cell).split(lineSeparator)
14      for node in nodes:
15         #--------------------------------------------------------------
16         # lines 19-23 find all the running servers belonging to the cell
17         # and node, and process them one at a time
18         #--------------------------------------------------------------
19         cname = AdminConfig.showAttribute(cell, 'name')
```

```
20         nname = AdminConfig.showAttribute(node, 'name')
21         servs = AdminControl.queryNames('type=Server,cell=' + cname + ',node=' + nname + ',*').
       split(lineSeparator)
22         print "Number of running servers on node " + nname + ": %s \n" % (len(servs))
23         for server in servs:
24           #---------------------------------------------------------
25           # lines 28-34 get some attributes from the server to display;
26           # invoke an operation on the server JVM to display a property.
27           #---------------------------------------------------------
28           sname = AdminControl.getAttribute(server, 'name')
29           ptype = AdminControl.getAttribute(server, 'processType')
30           pid   = AdminControl.getAttribute(server, 'pid')
31           state = AdminControl.getAttribute(server, 'state')
32            jvm = AdminControl.queryNames('type=JVM,cell=' +
 cname + ',node=' + nname + ',process=' + sname + ',*')
33           osname = AdminControl.invoke(jvm, 'getProperty', 'os.name')
34            print " " + sname + " " +  ptype + " has pid " + pid + "; state: " + state + "; on " +
 osname + "\n"
35
36            #---------------------------------------------------------
37            # line 40-45 find the applications running on this server and
38           # display the application name.
39           #---------------------------------------------------------
40           apps = AdminControl.queryNames('type=Application,cell=' +
 Cname + ',node=' + nname + ',process=' + sname + ',*').split(lineSeparator)
41           print "Number of applications running on " + sname + ": %s \n" % (len(apps))
42           for app in apps:
43              aname = AdminControl.getAttribute(app, 'name')
44              print aname + "\n"
45          print "----------------------------------------------------"
46          print "\n"
```

- Example output:

```
Number of running servers on node mynode: 2
mynode (NodeAgent) has pid 3592; state: STARTED; on Windows 2000
Number of applications running on mynode: 0
----------------------------------------------------

server1 (ManagedProcess) has pid 3972; state: STARTED; on Windows 2000
Number of applications running on server1: 0
----------------------------------------------------

Number of running servers on node mynodeManager: 1
dmgr (DeploymentManager) has pid 3308; state: STARTED; on Windows 2000
Number of applications running on dmgr: 2
adminconsole
filetransfer
----------------------------------------------------
```

Starting listener ports using scripting

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

Perform the following steps to start a listener port on an application server. The following example returns a list of listener port MBeans:

1. Identify the listener port MBeans for the application server and assign it to the lPorts variable.
   - Using Jacl:

     ```
     set lPorts [$AdminControl queryNames type=ListenerPort,cell=mycell,node=mynode,process=server1,*]
     ```

   - Using Jython:

     ```
     lPorts = AdminControl.queryNames('type=ListenerPort,cell=mycell,node=mynode,process=server1,*')
     print lPorts
     ```

   Example output:

```
WebSphere:cell=mycell,name=ListenerPort,mbeanIdentifier=server.xml#ListenerPort_1,type=ListenerPort,
  node=mynode,process=server1
WebSphere:cell=mycell,name=listenerPort,mbeanIdentifier=ListenerPort,type=server.xml#ListenerPort_2,
  node=mynode,process=server1
```

2. Start the listener port if it is not started. For example:
   - Using Jacl:

```
foreach lPort $lPorts {
     set state [$AdminControl getAttribute $lport started]
     if {$state == "false"} {
        $AdminControl invoke $lPort start
     }
  }
```

   - Using Jython:

```
# get line separator
import  java
lineSeparator = java.lang.System.getProperty('line.separator')

lPortsArray = lPorts.split(lineSeparator)
for lPort in lPortsArray:
 state = AdminControl.getAttribute(lPort, 'started')
 if state == 'false':
  AdminControl.invoke(lPort, 'start')
```

   These pieces of Jacl and Jython code loop through the listener port MBeans. For each listener port MBean, get the attribute value for the started attribute. If the attribute value is set to `false`, then start the listener port by invoking the start operation on the MBean.

Managing generic servers using scripting

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

A generic server is a server that the WebSphere Application Server manages but did not supply. You can use WebSphere Application Server to define, start, stop, and monitor generic servers.

- To define a generic server, use the following example:
  – Using Jacl:

```
$AdminTask createGenericServer mynode {-name generic1 -ConfigProcDef
{{"/usr/bin/myStartCommand" "arg1 arg2" "" "" "/tmp/workingDirectory" "/tmp/stopCommand" "argy argz"}}}
$AdminConfig save
```

  – Using Jython:

```
AdminTask.createGenericServer('mynode', '[-name generic1 -ConfigProcDef
[[c:\tmp\myStartCommand.exe "a b c" "" "" C:\tmp\myStopCommand "x y z"]]]')
AdminConfig.save()
```

- To start a generic server, use the `launchProcess` parameter, for example:
  – Using Jacl:

```
set nodeagent [$AdminControl queryNames *:*,type=NodeAgent]
$AdminControl invoke $nodeagent launchProcess generic1
```

  – Using Jython:

```
nodeagent = AdminControl.queryNames ('*:*,type=NodeAgent')
AdminControl.invoke(nodeagent, 'launchProcess', 'generic1')
```

  Example output:

```
true
```

  or

```
false
```

- To stop a generic server, use the `terminate` parameter, for example:
  – Using Jacl:

```
set nodeagent [$AdminControl queryNames *:*,type=NodeAgent]
$AdminControl invoke $nodeagent terminate generic1
```

  - Using Jython:

```
nodeagent = AdminControl.queryNames ('*:*,type=NodeAgent')
AdminControl.invoke(nodeagent, 'terminate', 'generic1')
```

  Example output:

```
true
```

  or

```
false
```

- To monitor the server state, use the `getProcessStatus` parameter, for example:
  - Using Jacl:

```
$AdminControl invoke $nodeagent getProcessStatus generic1
```

  Using Jython:

```
AdminControl.invoke(nodeagent, 'getProcessStatus', 'generic1')
```

  Example output:

```
RUNNING
```

  or

```
STOPPED
```

Setting development mode for server objects using scripting

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

Perform the following steps to set the development mode for a server object:

1. Locate the server object. The following example selects the first server found:
   - Using Jacl:

```
set server [$AdminConfig getid /Server:server1/]
```

   - Using Jython:

```
server = AdminConfig.getid('/Server:server1/')
```

2. Enable development mode:
   - Using Jacl:

```
$AdminConfig modify $server "{developmentMode true}"
```

   - Using Jython:

```
AdminConfig.modify(server, [['developmentMode', 'true']])
```

3. Save the configuration changes. See the “Saving configuration changes with the wsadmin tool” on page 232 article for more information.
4. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 article for more information.

Disabling parallel startup using scripting

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

Perform the following steps to disable parallel startup:

1. Locate the server object. The following example selects the first server found:
   - Using Jacl:

```
set server[$AdminConfig getid /Server:server1/]
```

   - Using Jython:

```
server = AdminConfig.getid('/Server:server1/']
```

2. Enable development mode. For example:
   - Using Jacl:

     ```
     $AdminConfig modify $server "{parallelStartEnabled false}"
     ```
   - Using Jython:

     ```
     AdminConfig.modify(server, [['parallelStartEnabled', 'false']])
     ```
3. Save the configuration changes. See the “Saving configuration changes with the wsadmin tool” on page 232 article for more information.
4. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 article for more information.

Removing multicast endpoints using scripting

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

WebSphere Application Server uses multicast broadcasting at the node level to allow a node agent to discover the managed processes in the node. The IPv4 and IPv6 multicast addresses are not compatible. Both the IPv4 and IPv6 multicast addresses are defined in the node agent configuration and when a node agent starts both addresses will be tried in sequence. It is recommended that you disable one of the multicast addresses after installation. By limiting multicast discovery to one protocol, the node agent will run more efficiently. Perform the following steps to remove a multicast endpoint:

1. Remove the multicast end point:
   - Using Jacl:

     ```
     set se [$AdminConfig getid /Node:y2001/ServerIndex:/]
     set eprs [lindex [$AdminConfig showAttribute $se endPointRefs] 0]
     foreach ep $eprs {
        set epName [$AdminConfig showAttribute $ep endPointName]
        if {$epName == "NODE_MULTICAST_DISCOVERY_ADDRESS"} {
           puts "Removing NODE_MULTICAST_DISCOVERY_ADDRESS..."
           $AdminConfig remove $ep
        }
     }
     ```
   - Using Jython:

     ```
     se = AdminConfig.getid('/Node:y2001/ServerIndex:/')
     import java
     lineseparator = java.lang.System.getProperty('line.separator')
     eprs = AdminConfig.showAttribute(se, ['endPointRefs']).split(lineseparator)[0]
     print eprs
     for ep in eprs:
        epName = AdminConfig.showAttribute(ep, ['endPointName'])
        if (epName) == "[NODE_MULTICAST_DISCOVERY_ADDRESS]":
           print "Removing NODE_MULTICAST_DISCOVERY_ADDRESS..."
           AdminConfig.remove(ep)
     ```
2. Save the configuration changes. See the “Saving configuration changes with the wsadmin tool” on page 232 article for more information.
3. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 article for more information.

Obtaining server version information with scripting

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

Perform the following steps to query the server version information:

1. Identify the server and assign it to the server variable.

- Using Jacl:

  ```
  set server [$AdminControl completeObjectName type=Server,name=server1,node=mynode,*]
  ```
- Using Jython:

  ```
  server = AdminControl.completeObjectName('type=Server,name=server1,node=mynode,*')
  print server
  ```

  Example output:

```
WebSphere:cell=mycell,name=server1,mbeanIdentifier=server.xml#Server_1,type=Server,node=mynode,
  process=server1,processType=ManagedProcess
```

2. Query the server version. The server version information is stored in the serverVersion attribute. The **getAttribute** command returns the attribute value of a single attribute, passing in the attribute name.
   - Using Jacl:

     ```
     $AdminControl getAttribute $server1 serverVersion
     ```
   - Using Jython:

     ```
     print AdminControl.getAttribute(server1, 'serverVersion')
     ```

   Example output for a Network Deployment installation follows:

```
IBM WebSphere Application Server Version Report
   ---------------------------------------------------------------------------

        Platform Information
        ------------------------------------------------------------------------

                Name: IBM WebSphere Application Server
                Version: 5.0


        Product Information
        ------------------------------------------------------------------------

                ID: BASE
                Name: IBM WebSphere Application Server
                Build Date: 9/11/02
                Build Level: r0236.11
                Version: 5.0.0


        Product Information
        ------------------------------------------------------------------------

                ID: ND
                Name: IBM WebSphere Application Server for Network Deployment
                Build Date: 9/11/02
                Build Level: r0236.11
                Version: 5.0.0

   ---------------------------------------------------------------------------
   End Report
   ---------------------------------------------------------------------------
```

Configuring security with scripting

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 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 Remote Method Invocation (RMI) connector, set the following properties in the `sas.client.props` file with the appropriate values:

    ```
    com.ibm.CORBA.loginUserid=
    com.ibm.CORBA.loginPassword=
    ```

    Also, set the following property:

    ```
    com.ibm.CORBA.loginSource=properties
    ```

    The default value for this property is `prompt` in the `sas.client.props` file. If you leave the default value, a dialog box appears with a password prompt. If the script is running unattended, it appears to hang.
  – If you use a Simple Object Access Protocol (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=
    ```
- 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.

  Warning: 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 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” on page 250 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” on page 250 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” on page 232 article for more information.
4. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 article for more information.

Configuring data access with scripting

This topic contains the following tasks:

- “Configuring a JDBC provider using scripting” on page 313
- “Configuring new data sources using scripting” on page 314
- “Configuring new connection pools using scripting” on page 315
- “Configuring new data source custom properties using scripting” on page 315
- “Configuring new J2CAuthentication data entries using scripting” on page 316

- “Configuring new WAS40 data sources using scripting” on page 317
- “Configuring new WAS40 connection pools using scripting” on page 318
- “Configuring new WAS40 custom properties using scripting” on page 319
- “Configuring new J2C resource adapters using scripting” on page 320
- “Configuring custom properties for J2C resource adapters using scripting” on page 321
- “Configuring new J2C connection factories using scripting” on page 322
- “Configuring new J2C authentication data entries using scripting” on page 324
- “Configuring new J2C administrative objects using scripting” on page 326
- “Configuring new J2C activation specs using scripting” on page 325
- “Testing data source connections using scripting” on page 328

Configuring a JDBC provider using scripting

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 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" on page 232 article for more information.
6. In a network deployment environment only, synchronize the node. See the "Synchronizing nodes with the wsadmin tool" on page 217 article for more information.

Configuring new data sources using scripting

Before starting this task, the wsadmin tool must be running. See the "Starting the wsadmin scripting client" on page 250 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” on page 232 article for more information.
6. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 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” on page 250 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” on page 232 article for more information.
4. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 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” on page 250 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” on page 232 article for more information.
7. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 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” on page 250 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” on page 232 article for more information.
6. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 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” on page 250 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” on page 232 article for more information.
6. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 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” on page 250 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” on page 232 article for more information.
6. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 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” on page 250 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” on page 232 article for more information.
7. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 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” on page 250 article for more information.

Perform the following steps to configure a new J2C resource adapter:

1. Identify the parent ID and assign it to the node variable. The following example uses the node configuration object as the parent. You can modify this example to use the cell, cluster, server, or application configuration object as the parent.
   - Using Jacl:

     ```
     set node [$AdminConfig getid /Cell:mycell/Node:mynode/]
     ```

   - Using Jython:

     ```
     node = AdminConfig.getid('/Cell:mycell/Node:mynode/')
     print node
     ```

   Example output:

   ```
   mynode(cells/mycell/nodes/mynode|node.xml#Node_1)
   ```

2. Identify the required attributes:

- Using Jacl:

  ```
  $AdminConfig required J2CResourceAdapter
  ```

- Using Jython:

  ```
  print AdminConfig.required('J2CResourceAdapter')
  ```

Example output:

```
Attribute     Type
name         String
```

3. Set up the required attributes:
   - Using Jacl:

     ```
     set rarFile c:/currentScript/cicseci.rar
     set option  [list  -rar.name  RAR1]
     ```

   - Using Jython:

     ```
     rarFile = 'c:/currentScript/cicseci.rar'
     option  = '[-rar.name  RAR1]'
     ```

4. Create a resource adapter:
   - Using Jacl:

     ```
     set newra [$AdminConfig installResourceAdapter $rarFile mynode $option]
     ```

   - Using Jython:

     ```
     newra = AdminConfig.installResourceAdapter(rarFile, 'mynode', option)
     print newra
     ```

   Example output:

   ```
   RAR1(cells/mycell/nodes/mynode|resources.xml#J2CResourceAdapter_1)
   ```

5. Save the configuration changes. See the “Saving configuration changes with the wsadmin tool” on page 232 article for more information.
6. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 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” on page 250 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” on page 232 article for more information.
7. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 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” on page 250 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” on page 232 article for more information.

4. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 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” on page 250 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” on page 232 article for more information.
6. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 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" on page 250 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" on page 232 article for more information.
4. In a network deployment environment only, synchronize the node. See the "Synchronizing nodes with the wsadmin tool" on page 217 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" on page 250 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” on page 232 article for more information.

4. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 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” on page 250 article for more information.

Perform the following steps to test a data source to ensure a connection to the database.

1. Identify the DataSourceCfgHelper MBean and assign it to the dshelper variable.
   - Using Jacl:

     ```
     set ds [$AdminConfig getid /DataSource:DS1/]
     $AdminControl testConnection $ds
     ```
   - Using Jython:

     ```
     ds = AdminConfig.getid('/DataSource:DS1/')
     AdminControl.testConnection(ds)
     ```

   Example output:

   ```
   WASX7217I: Connection to provided datasource was successful.
   ```
2. Test the connection. The following example invokes the testConnectionToDataSource operation on the MBean, passing in the classname, userid, password, database name, JDBC driver class path, language, and country.
   - Using Jacl:

     ```
     $AdminControl invoke $dshelper testConnectionToDataSource "COM.ibm.db2.jdbc.DB2XADataSource db2admin db2admin
     {{databaseName sample}} c:/sqllib/java/db2java.zip en US"
     ```
   - Using Jython:

     ```
     print AdminControl.invoke(dshelper, 'testConnectionToDataSource',
     'COM.ibm.db2.jdbc.DB2XADataSource dbuser1 dbpwd1
       "{{databaseName jtest1}}" c:/sqllib/java12/db \"\" \"\"')
     ```

   Example output:

   ```
   WASX7217I: Connection to provided data source was successful.
   ```

Configuring messaging with scripting

This topic contains the following tasks:

- “Configuring the message listener service using scripting”
- “Configuring new JMS providers using scripting” on page 330
- “Configuring new JMS destinations using scripting” on page 331
- “Configuring new JMS connections using scripting” on page 332
- “Configuring new WebSphere queue connection factories using scripting” on page 333
- “Configuring new WebSphere topic connection factories using scripting” on page 334
- “Configuring new WebSphere queues using scripting” on page 335
- “Configuring new WebSphere topics using scripting” on page 336
- “Configuring new MQ queue connection factories using scripting” on page 337
- “Configuring new MQ topic connection factories using scripting” on page 338
- “Configuring new MQ queues using scripting” on page 339
- “Configuring new MQ topics using scripting” on page 340

Configuring the message listener service using scripting

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 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” on page 232 article for more information.

5. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 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” on page 250 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” on page 232 article for more information.

6. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 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” on page 250 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” on page 232 article for more information.
6. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 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” on page 250 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” on page 232 article for more information.
6. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 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” on page 250 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” on page 232 article for more information.
6. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 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” on page 250 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” on page 232 article for more information.
6. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 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” on page 250 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” on page 232 article for more information.
6. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 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” on page 250 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” on page 232 article for more information.
6. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 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” on page 250 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” on page 232 article for more information.
7. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 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” on page 250 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” on page 232 article for more information.
7. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 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” on page 250 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” on page 232 article for more information.
7. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 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” on page 250 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” on page 232 article for more information.
6. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 article for more information.

Configuring mail, URLs, and resource environment entries with scripting

This topic contains the following tasks:

- “Configuring new mail providers using scripting” on page 342
- “Configuring new mail sessions using scripting” on page 342
- “Configuring new protocols using scripting” on page 343
- “Configuring new custom properties using scripting” on page 344
- “Configuring new resource environment providers using scripting” on page 345
- “Configuring custom properties for resource environment providers using scripting” on page 346
- “Configuring new referenceables using scripting” on page 347
- “Configuring new resource environment entries using scripting” on page 348
- “Configuring custom properties for resource environment entries using scripting” on page 349
- “Configuring new URL providers using scripting” on page 350
- “Configuring custom properties for URL providers using scripting” on page 351
- “Configuring new URLs using scripting” on page 352
- “Configuring custom properties for URLs using scripting” on page 353

Configuring new mail providers using scripting

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 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” on page 232 article for more information.
6. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 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” on page 250 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” on page 232 article for more information.
6. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 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” on page 250 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” on page 232 article for more information.
6. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 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” on page 250 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” on page 232 article for more information.
7. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 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” on page 250 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” on page 232 article for more information.
6. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 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” on page 250 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” on page 232 article for more information.
7. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 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” on page 250 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” on page 232 article for more information.
6. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 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” on page 250 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” on page 232 article for more information.
6. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 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” on page 250 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” on page 232 article for more information.
7. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 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” on page 250 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” on page 232 article for more information.
6. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 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” on page 250 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” on page 232 article for more information.
7. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 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” on page 250 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” on page 232 article for more information.
6. In a network deployment environment only, synchronize the node. See the “Synchronizing nodes with the wsadmin tool” on page 217 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” on page 250 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" on page 232 article for more information.
7. In a network deployment environment only, synchronize the node. See the "Synchronizing nodes with the wsadmin tool" on page 217 article for more information.

Troubleshooting with scripting

This topic contains the following tasks:

- "Tracing operations with the wsadmin tool"
- "Configuring traces using scripting" on page 355
- "Turning traces on and off in servers processes using scripting" on page 356
- "Dumping threads in server processes using scripting" on page 357
- "Setting up profile scripts to make tracing easier using scripting" on page 357

Tracing operations with the wsadmin tool

Before starting this task, the wsadmin tool must be running. See the "Starting the wsadmin scripting client" on page 250 article for more information.

Perform the following steps to trace operations:

Enable wsadmin client tracing with the following command:

- Using Jacl:
  ```
  $AdminControl trace com.ibm.*=all=enabled
  ```
- Using Jython:
  ```
  AdminControl.trace('com.ibm.*=all=enabled')
  ```

where:

$	is a Jacl operator for substituting a variable name with its value
AdminControl	is an object that enables the manipulation of MBeans running in a WebSphere server process
trace	is an AdminControl command
com.ibm.*=all=enabled	indicates to turn on tracing

The following command disables tracing:

- Using Jacl:

```
$AdminControl trace com.ibm.*=all=disabled
```

- Using Jython:

```
AdminControl.trace('com.ibm.*=all=disabled')
```

where:

$	is a Jacl operator for substituting a variable name with its value
AdminControl	is an object that enables the manipulation of MBeans running in a WebSphere server process
trace	is an AdminControl command
com.ibm.*=all=disabled	indicates to turn off tracing

The trace command changes the trace settings for the current session. You can change this setting persistently by editting the wsadmin.properties file. The property com.ibm.ws.scripting.traceString is read by the launcher during initialization. If it has a value, the value is used to set the trace.

A related property, com.ibm.ws.scripting.traceFile, designates a file to receive all trace and logging information. The wsadmin.properties file contains a value for this property. Run the wsadmin tool with a value set for this property. It is possible to run without this property set, where all logging and tracing goes to the administrative console.

Configuring traces using scripting

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

Perform the following steps to set the trace for a configured server:

1. Identify the 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 trace service belonging to the server and assign it to the tc variable:
   - Using Jacl:

     ```
     set tc [$AdminConfig list TraceService $server]
     ```

   - Using Jython:

     ```
     tc = AdminConfig.list('TraceService', server)
     print tc
     ```

Example output:

```
(cells/mycell/nodes/mynode/servers/server1|server.xml#TraceService_1)
```

3. Set the trace string. The following example sets the trace string for a single component:
   - Using Jacl:
     ```
     $AdminConfig modify $tc {{startupTraceSpecification com.ibm.websphere.management.*=all=enabled}}
     ```
   - Using Jython:
     ```
     AdminConfig.modify(tc, [['startupTraceSpecification',  'com.ibm.websphere.management.*=all=enabled']])
     ```
4. The following command sets the trace string for multiple components:
   - Using Jacl:
     ```
     $AdminConfig modify $tc {{startupTraceSpecification com.ibm.websphere.management.*=all=
       enabled:com.ibm.ws.management.*=all=enabled:com.ibm.ws.runtime.*=all=enabled}}
     ```
   - Using Jython:
     ```
     AdminConfig.modify(tc, [['startupTraceSpecification',  'com.ibm.websphere.management.
       *=all=enabled:com.ibm.ws.management.*=all=
       enabled:com.ibm.ws.runtime.*=all=enabled']])
     ```
5. Save the configuration changes. See the "Saving configuration changes with the wsadmin tool" on page 232 article for more information.
6. In a network deployment environment only, synchronize the node. See the "Synchronizing nodes with the wsadmin tool" on page 217 article for more information.

Turning traces on and off in servers processes using scripting

Before starting this task, the wsadmin tool must be running. See the "Starting the wsadmin scripting client" on page 250 article for more information.

Perform the following steps to turning traces on and off in server processes:

1. Identify the object name for the TraceService MBean running in the process:
   - Using Jacl:
     ```
     $AdminControl completeObjectName type=TraceService,node=mynode,process=server1,*
     ```
   - Using Jython:
     ```
     AdminControl.completeObjectName('type=TraceService,node=mynode,process=server1,*')
     ```
2. Obtain the name of the object and set it to a variable:
   - Using Jacl:
     ```
     set ts [$AdminControl completeObjectName type=TraceService,process=server1,*]
     ```
   - Using Jython:
     ```
     ts = AdminControl.completeObjectName('type=TraceService,process=server1,*')
     ```
3. Turn tracing on or off for the server. For example:
   - To turn tracing on, perform the following step:
     - Using Jacl:
       ```
       $AdminControl setAttribute $ts traceSpecification com.ibm.*=all=enabled
       ```
     - Using Jython:
       ```
       AdminControl.setAttribute(ts, 'traceSpecification', 'com.ibm.*=all=enabled')
       ```
   - To turn tracing off, perform the following step:
     - Using Jacl:
       ```
       $AdminControl setAttribute $ts traceSpecification com.ibm.*=all=disabled
       ```
     - Using Jython:
       ```
       AdminControl.setAttribute(ts, 'traceSpecification', 'com.ibm.*=all=disabled')
       ```

Dumping threads in server processes using scripting

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

Use the AdminControl object to dump the Java threads of a running server. The following example produces a Java core file. You can use this file for problem determination.

- Using Jacl:

```
set jvm [$AdminControl completeObjectName type=JVM,process=server1,*]
$AdminControl invoke $jvm dumpThreads
```

- Using Jython:

```
jvm = AdminControl.completeObjectName('type=JVM,process=server1,*')
AdminControl.invoke(jvm, 'dumpThreads')
```

Setting up profile scripts to make tracing easier using scripting

Before starting this task, the wsadmin tool must be running. See the “Starting the wsadmin scripting client” on page 250 article for more information.

Set up a profile script to make tracing easier. The following profile script example turns tracing on and off for `server1`:

- Using Jacl:

```
proc ton {} {
  global AdminControl
  set ts [$AdminControl queryNames type=TraceService,node=mynode,process=server1,*]
  $AdminControl setAttribute $ts traceSpecification com.ibm.=all=enabled
}

proc toff {} {
  global AdminControl
  set ts [$AdminControl queryNames type=TraceService,node=mynode,process=server1,*]
  $AdminControl setAttribute $ts traceSpecification com.ibm.*=all=disabled
}

proc dt {} {
  global AdminControl
  set jvm [$AdminControl queryNames type=JVM,node=mynode,process=server1,*]
  $AdminControl invoke $jvm dumpThreads
}
```

- Using Jython:

```
def ton():
       global lineSeparator
       ts = AdminControl.queryNames('type=TraceService,node=mynode,process=server1,*')

       AdminControl.setAttribute(ts, 'traceSpecification', 'com.ibm.=all=enabled')

def toff():
       global lineSeparator
       ts = AdminControl.queryNames('type=TraceService,node=mynode,process=server1,*')

       AdminControl.setAttribute(ts, 'traceSpecification', 'com.ibm.*=all=disabled')

def dt():
       global lineSeparator
       jvm = AdminControl.queryNames('type=JVM,node=mynode,process=server1,*')
       AdminControl.invoke(jvm, 'dumpThreads')
```

If you start the wsadmin tool with this profile script, you can use the **ton** command to turn on tracing in the server, the **toff** command to turn off tracing, and the **dt** command to dump the Java threads. For more information about running scripting commands in a profile script, see the “Starting the wsadmin scripting client” on page 250 article.

Scripting reference material

This topic contains the following tasks:

- “Wsadmin tool”
- “Commands for the AdminConfig object” on page 375
- “Commands for the AdminControl object” on page 396
- “Commands for the AdminApp object” on page 415
- “Commands for the AdminTask object” on page 489
- “Administrative command invocation syntax” on page 623
- “Commands for the Help object” on page 362
- “Properties used by scripted administration” on page 624

Wsadmin tool

The WebSphere Application Server wsadmin tool runs scripts. You can use the wsadmin tool to manage WebSphere Application Server as well as the configuration, application deployment, and server run-time operations.

The command-line invocation syntax for the wsadmin scripting client is as follows:

```
wsadmin [-h(help)]

[-?]

[-c <commands>]

[-p <properties_file_name>]

[-profile <profile_script_name>]

[-profileName <profile_name>]

[-f <script_file_name>]

[-javaoption java_option]

[-lang language]

[-wsadmin_classpath classpath]

[-conntype SOAP [-host host_name] [-port port_number] [-user userid] [-password password] |

     RMI [-host host_name] [-port port_number] [-user userid] [-password password] |

     NONE
]

[script parameters]
```

Where *script parameters* represent any arguments other than the ones listed previously. The argc variable contains the argument number, and the argv variable contains the contents.

Options

-c Designates to run a single command.

Multiple -c options can exist on the command line. They run in the order that you designate. You must save after using this command.

-f Designates a script to run.

Only one -f option can exist on the command line.

-javaoption
Specifies a valid Java standard or a non-standard option.

Multiple -javaoption options can exist on the command line.

-lang
Specifies the language of the script file, the command, or an interactive shell. The possible languages include: Jacl and Jython. The options for the -lang argument include: jacl and jython.

This option overrides language determinations that are based on a script file name, or the com.ibm.ws.scripting.defaultLang property. The `-lang` argument has no default value. If the command line or the property does not supply the script language, and the wsadmin tool cannot determine it, an error message generates. This argument is required if not determined from the script file name.

-p
Specifies a properties file.

The file listed after -p, represents a Java properties file that the scripting process reads. Three levels of default properties files load before the properties file that you specify on the command line. The first level is the installation default, `wsadmin.properties`, which is located in the WebSphere Application Server `properties` directory. The second level is the user default, `wsadmin.properties`, which is located in your `home` directory. The third level is the properties file that the environment variable `WSADMIN_PROPERTIES` points to.

Multiple -p options can exist on the command line. They invoke in the order that you supply them.

-profile
Specifies a profile script.

The profile script runs before other commands, or scripts. If you specify `-c`, the profile script runs before it invokes this command. If you specify `-f`, the profile script runs before it runs the script. In interactive mode, you can use the profile script to perform any standard initialization that you want. You can specify multiple -profile options on the command line, and they invoke in the order that you supply them.

-profileName
Specifies the profile from which the wsadmin tool will run. Specify this option if one the following reasons apply:

- You run the wsadmin tool from the `WAS_HOME/bin` directory and you do not have a default profile or you want to run in a profile other than the default profile.
- You are currently in a profile `bin` directory but want to run the wsadmin tool from a different profile.

-?
Provides syntax help.

-help
Provides syntax help.

-conntype
Specifies the type of connection to use.

This argument consists of a string that determines the type, for example, SOAP, and the options that are specific to that connection type. Possible types include: `SOAP`, `RMI`, and `NONE`.

Use the -conntype NONE option to run in local mode. The result is that the scripting client is not connected to a running server. You can manage server configuration, the installation and the uninstallation of applications without the application server running.

-wsadmin_classpath

Use this option to make additional classes available to your scripting process.

Follow this option with a class path string. For example:

```
c:/MyDir/Myjar.jar;d:/yourdir/yourdir.jar
```

The class path is then added to the class loader for the scripting process.

You can also specify this option in a properties file that is used by the wsadmin tool. The property is com.ibm.ws.scripting.classpath. If you specify -wsadmin_classpath on the command line, the value of this property overrides any value that is specified in a properties file. The class path property and the command-line options are not concatenated.

-host

Specify a hostname to which wsadmin should attempt to connect. The default wsadmin.properties file located in the properties directory of each WebSphere profile provides localhost as the value of the host property if this option is not specified.

-password

Specify a password to be used by the connector to connect to the server if security is enabled in the server.

Warning: 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 profile.

-username

Specify a user name to be used by the connector to connect to the server if security is enabled in the server.

-port

Specify a port to be used by the connector. The default wsadmin.properties file located in the properties directory of each WebSphere Application Server profile provides a value in the port property to connect to the local server.

In the following syntax examples, *mymachine* is the name of the host in the `wsadmin.properties` file that is specified by the com.ibm.ws.scripting.port property:

SOAP connection to the local host

Use the options that are defined in the `wsadmin.properties` file.

SOAP connection to the *mymachine* host

Using Jacl:

```
wsadmin -f test1.jacl -profile setup.jacl -conntype SOAP -port mymachinesoapportnumber -host mymachine
```

Using Jython:

```
wsadmin -lang jython -f  test1.py -profile setup.py -conntype SOAP -port mymachinesoapportnumber -host mymachine
```

Initial and maximum Java heap size

Using Jacl:

```
wsadmin -javaoption -Xms128m -javaoption -Xmx256m -f test.jacl
```

Using Jython:

```
wsadmin -lang jython -javaoption -Xms128m -javaoption -Xmx256m -f test.py
```

RMI connection with security

Using Jacl:

```
wsadmin -conntype RMI -port  rmiportnumber -userid userid -password password
```

Using Jython:

```
wsadmin -lang jython -conntype RMI -port  rmiportnumber -userid userid -password password
```

Warning: 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 profile.

Local mode of operation to perform a single command

Using Jacl:

```
wsadmin -conntype NONE -c "$AdminApp uninstall app"
```

or

Using Jython:

```
wsadmin -lang jython -conntype NONE -c "AdminApp.uninstall('app')"
```

or

wsadmin tool performance tips: The following performance tips are for the wsadmin tool:

- If the deployment manager is running at a higher service maintenance level than that of the node agent, you must run the `wsadmin.sh` or the `wsadmin.bat` from the `bin` directory of the deployment manager.
- When you launch a script using the `wsadmin.bat` or the `wsadmin.sh` files, a new process is created with a new Java virtual machine (JVM) API. If you use scripting with multiple wsadmin **-c** commands from a batch file or a shell script, these commands run slower than if you use a single wsadmin **-f** command. The -f option runs faster because only one process and JVM API are created for installation and the Java classes for the installation load only once.

  The following example, illustrates running multiple application installation commands from a batch file:

  Using Jacl:

  ```
  wsadmin -c "$AdminApp install c:\\myApps\\App1.ear {-appname appl1}"
  wsadmin -c "$AdminApp install c:\\myApps\\App2.ear {-appname appl2}"
  wsadmin -c "$AdminApp install c:\\myApps\\App3.ear {-appname appl3}"
  ```

  or

  Using Jython:

  ```
  wsadmin -lang jython -c "AdminApp.install('c:\myApps\App1.ear', '[-appname appl1]')"
  wsadmin -lang jython -c "AdminApp.install('c:\myApps\App2.ear', '[-appname appl2]')"
  wsadmin -lang jython -c "AdminApp.install('c:\myApps\App3.ear', '[-appname appl3]')"
  ```

  or

  Or, for example, using Jacl, you can create the *appinst.jacl* file that contains the commands:

  ```
  $AdminApp install c:\\myApps\\App1.ear {-appname appl1}
  $AdminApp install c:\\myApps\\App2.ear {-appname appl2}
  $AdminApp install c:\\myApps\\App3.ear {-appname appl3}
  ```

  Invoke this file using the following command: `wsadmin -f` *`appinst.jacl`*

  Or using Jython, you can create the *appinst.py* file, that contains the commands:

  ```
  AdminApp.install('c:\myApps\App1.ear', '[-appname appl1]')
  AdminApp.install('c:\myApps\App2.ear', '[-appname appl2]')
  AdminApp.install('c:\myApps\App3.ear', '[-appname appl3]')
  ```

  Then invoke this file using the following command: `wsadmin -lang jython -f` *`appinst.py`*.
- Use the AdminControl **queryNames** and **completeObjectName** commands carefully with a large installation. For example, if only a few beans exist on a single machine, the `$AdminControl queryNames *` command performs well. If a scripting client connects to the deployment manager in a multiple machine environment, use a command only if it is necessary for the script to obtain a list of all the

MBeans in the system. If you need the MBeans on a node, it is easier to invoke "$AdminControl queryNames node=mynode,*". The JMX system management infrastructure forwards requests to the system to fulfill the first query, *. The second query, node=mynode,* is targeted to a specific machine.

- The WebSphere Application Server is a distributed system, and scripts perform better if you minimize remote requests. If some action or interrogation is required on several items, for example, servers, it is more efficient to obtain the list of items once and iterate locally. This procedure applies to the actions that the AdminControl object performs on running MBeans, and actions that the AdminConfig object performs on configuration objects.

Commands for the Help object

The Help object provides general help and dynamic online information about the currently running MBeans. You can use the Help object as an aid in writing and running scripts with the AdminControl object.

The following commands are available for the Help object:

Command name:	Description:	Parameters and return values:	Examples:

AdminApp	Provides a summary of all of the available methods for the AdminApp object.	• Parameters: none
		• Returns: string
			Example usage:
			Using Jacl:
			`$Help AdminApp`
			Using Jython:
			`print Help.AdminApp()`
			Example output:
			`WASX7095I: The AdminApp object`
			`allows application objects to`
			`be manipulated -- this includes`
			`installing, uninstalling, editing,`
			`and listing.  Most of the commands`
			`supported by AdminApp operate in two`
			`modes: the default mode is one`
			`in which AdminApp communicates with the`
			`WebSphere Application Server to accomplish`
			`its tasks.  A local mode is also`
			`possible, in which no server`
			`communication takes place.  The local`
			`mode of operation is invoked`
			`by bringing up the scripting client with`
			`no server connected using the command line`
			`"-conntype NONE" option or setting the`
			`"com.ibm.ws.scripting.connectionType=NONE"`
			`property in the wsadmin.properties.`
			
			`The following commands are supported`
			`by AdminApp; more detailed information`
			`about each of these commands`
			`is available by using the`
			`"help" command of AdminApp and`
			`supplying the name of the command`
			`as an argument.`
			
			`deleteUserAndGroupEntries`
			`Deletes all the user/group`
			`information for all the roles and all`
			`the username/password information for RunAs`
			`roles for a given application.`
			
			`edit Edit the properties of an application`
			
			`editInteractive Edit the properties`
			`of an application interactively`
			
			`export Export application to a file`
			
			`exportDDL Export DDL from`
			`application to a directory`
			
			`help Show help information`
			
			`install Installs an application,`
			`given a file name and an option string.`
			
			`installInteractive Installs an application`
			`in interactive mode, given a file name`
			`and an option string.`
			
			`list List all installed applications`

AdminApp continued			`listModules List the modules in a specified application`
			`options Shows the options available, either for a given file, or in general.`
			`publishWSDL Publish WSDL files for a given application`
			`taskInfo Shows detailed information pertaining to a given installation task for a given file`
			`uninstall Uninstalls an application, given an application name and an option string`
			`updateAccessIDs Updates the user/group binding information with accessID from user registry for a given application`
			`view View an application or module, given an application or module name`

AdminConfig	Provides a summary of all the available methods for the AdminConfig object.	• Parameters: None
		• Returns: string
			Example usage:
			Using Jacl:
			`$Help AdminConfig`
			Using Jython:
			`print Help.AdminConfig()`
			Example output:
			`WASX7053I: The following functions are supported by AdminConfig:`
			`create Creates a configuration object, given a type, a parent, and a list of attributes`
			`create Creates a configuration object, given a type, a parent, a list of attributes, and an attribute name for the new object`
			`remove Removes the specified configuration object`
			`list Lists all configuration objects of a given type`
			`list Lists all configuration objects of a given type, contained within the scope supplied`
			`show Show all the attributes of a given configuration object`
			`show Show specified attributes of a given configuration object`
			`modify Change specified attributes of a given configuration object`
			`getId Show the configId of an object, given a string version of its containment`
			`contents Show the objects which a given type contains`
			`parents Show the objects which contain a given type`
			`attributes Show the attributes for a given type`
			`types Show the possible types for configuration`
			`help Show help information`

AdminControl	Provides a summary of the help commands and ways to invoke an administrative command.	• Parameters: None
		• Returns: string
			Example usage:
			Using Jacl:
			`$Help AdminControl`
			Using Jython:
			`print Help.AdminControl()`
			Example output:
			`WASX7027I: The following functions are supported by AdminControl:`
			`getHost returns String representation of connected host`
			`getPort returns String representation of port in use`
			`getType returns String representation of connection type in use`
			`reconnect reconnects with server`
			`queryNames Given ObjectName and QueryExp, retrieves set of ObjectNames that match.`
			`queryNames Given String version of ObjectName, retrieves String of ObjectNames that match.`
			`getMBeanCount returns number of registered beans`
			`getDomainName returns "WebSphere"`
			`getDefaultDomain returns "WebSphere"`
			`getMBeanInfo Given ObjectName, returns MBeanInfo structure for MBean`
			`isInstanceOf Given ObjectName and class name, true if MBean is of that class`
			`isRegistered true if supplied ObjectName is registered`
			`isRegistered true if supplied String version of ObjectName is registered`
			`getAttribute Given ObjectName and name of attribute, returns value of attribute`
			`getAttribute Given String version of ObjectName and name of attribute, returns value of attribute`
			`getAttributes Given ObjectName and array of attribute names, returns AttributeList`
			`getAttributes Given String version of ObjectName and attribute names, returns String of name value pairs`

			setAttribute Given ObjectName and Attribute object, set attribute for MBean specified setAttribute Given String version of ObjectName, attribute name and attribute value, set attribute for MBean specified setAttributes Given ObjectName and AttributeList object, set attributes for the MBean specified invoke Given ObjectName, name of method, array of parameters and signature, invoke method on MBean specified invoke Given String version of ObjectName, name of method, String version of parameter list, invoke method on MBean specified. invoke Given String version of ObjectName, name of method, String version of parameter list, and String version of array of signatures, invoke method on MBean specified. makeObjectName Return an ObjectName built with the given string completeObjectName Return a String version of an object name given a template name trace Set the wsadmin trace specification help Show help information

<table>
<tr>
<td>AdminTask</td>
<td>Provides a summary of help commands and ways to invoke an administrative command.</td>
<td>

- Parameters: None
- Returns: string

</td>
<td>
Example usage:

Using Jacl:

```
$AdminTask help
```

Using Jython:

```
print AdminTask.help()
```

Example output:

```
WASX8001I: The AdminTask object enables the
 available administrative commands.
AdminTask commands operate in two modes:
the default mode is one which AdminTask
communicates with the WebSphere Application
 Server to accomplish its task. A local mode
is also available in which no server
communication takes place. The local mode of
operation is invoked by bringing up the
scripting client busing the command line
"-conntype NONE" option or setting the
"com.ibm.ws.scripting.connectiontype=NONE"
 property in wsadmin.properties file.

The number of administrative commands varies
and depends on your WebSphere Application
Server installation. Use the following help
commands to obtain a list of supported
commands and their parameters:

help -commands      list all the
 administrative commands
help -commandGroups    list all the
 administrative command groups
help commandName      display detailed
 information for the specified command
help commandName stepName display detailed
 information for the specified step
 belonging to the specified command
help commandGroupName   display detailed
 information for the specified command group

There are various flavors to invoke an
administrative  command. They are

commandName invokes an administrative
 command that does not require any argument.

commandName targetObject invokes an admin
 command with the target object string,
 for example the configuration object name
 of a resource adapter. The expected target
 object varies with the administrative
 command invoked. Use help command to get
 information on the target object of an
 administrative command.

commandName  options invokes an administrative
 command with the specified option strings.
 This invocation syntax is used to invoke
 an administrative command that does not
 require a target object. It is also used
 to enter interactive mode if "-interactive"
 mode is included in the options string.
```
</td>
</tr>
</table>

AdminTask continued			`commandName targetObject options invokes an administrative command with the specified target object and options strings. If "-interactive" is included in the options string, then interactive mode is entered. The target object and options strings vary depending on the admin command invoked. Use help command to get information on the target object and options.`

all	Provides a summary of the information that the MBean defines by name.	• Parameters: name - string
		• Returns: string
			Example usage:
			Using Jacl:
			`$Help all [$AdminControl queryNames type`
			`=TraceService,process=server1,node=pongo,*]`
			Using Jython:
			`print Help.all(AdminControl.queryNames`
			`('type=TraceService,process=server1,`
			`node=pongo,*'))`
			Example output:
			`Name: WebSphere:cell=pongo,name=`
			`TraceService,mbeanIdentifier=cells/`
			`pongo/nodes/pongo/servers/server1/`
			`server.xml#TraceService_1,`
			`type=TraceService,node=pongo,process=server1`
			`Description: null`
			`Class name:`
			`javax.management.modelmbean.RequiredModelMBean`
			
			`Attribute         Type      Access`
			`ringBufferSize       int       RW`
			`traceSpecification   java.lang.String   RW`
			
			`Operation`
			`int getRingBufferSize()`
			`void setRingBufferSize(int)`
			`java.lang.String getTraceSpecification()`
			`void setTraceState(java.lang.String)`
			`void appendTraceString(java.lang.String)`
			`void dumpRingBuffer(java.lang.String)`
			`void clearRingBuffer()`
			`[Ljava.lang.String;`
			`listAllRegisteredComponents()`
			`[Ljava.lang.String;`
			`listAllRegisteredGroups()`
			`[Ljava.lang.String;`
			`listComponentsInGroup(java.lang.String)`
			`[Lcom.ibm.websphere.ras.TraceElementState;`
			`getTracedComponents()`
			`[Lcom.ibm.websphere.ras.TraceElementState;`
			`getTracedGroups()`
			`java.lang.String getTraceSpecification`
			`   (java.lang.String)`
			`void processDumpString(java.lang.String)`
			`void checkTraceString(java.lang.String)`
			`void setTraceOutputToFile(java.lang.String,`
			`int, int, java.lang.String)`
			`void setTraceOutputToRingBuffer(int,`
			`java.lang.String)`
			`java.lang.String rolloverLogFileImmediate`
			`(java.lang.String, java.lang.String)`
			
			`Notifications`
			`jmx.attribute.changed`
			
			`Constructors`

attributes	Provides a summary of all the attributes that the MBean defines by name.	- Parameters: name - string
		- Returns: string
			Example usage:
			Using Jacl:
			`$Help attributes [$AdminControl queryNames type=TraceService,process=server1,node=pongo,*]`
			Using Jython:
			`print Help.attributes (AdminControl.queryNames('type=TraceService, process=server1,node=pongo,*'))`
			Example output:
			`Attribute Type Access`
			`ringBufferSize java.lang.Integer RW`
			`traceSpecification string RW`

classname	Provides a class name that the MBean defines by name.	- Parameters: name - string
		- Returns: string
			Example usage:
			Using Jacl:
			`$Help classname [$AdminControl queryNames type=TraceService,process=server1,node=pongo,*]`
			Using Jython:
			`print Help.classname(AdminControl.queryNames ('type=TraceService,process=server1,node=pongo,*'))`
			Example output:
			`javax.management.modelmbean.RequiredModelMBean`

constructors	Provides a summary of all of the constructors that the MBean defines by name.	- Parameters: name - string
		- Returns: string
			Example usage:
			Using Jacl:
			`$Help constructors [$AdminControl queryNames type=TraceService,process=server1,node=pongo,*]`
			Using Jython:
			`print Help.constructors (AdminControl.queryNames('type=TraceService, process=server1,node=pongo,*'))`
			Example output:
			`Constructors`

description	Provides a description that the MBean defines by name.	- Parameters: name - string
		- Returns: string
			Example usage:
			Using Jacl:
			`$Help description [$AdminControl queryNames type=TraceService,process=server1,node=pongo,*]`
			Using Jython:
			`print Help.description(AdminControl.queryNames ('type=TraceService,process=server1,node=pongo,*'))`
			Example output:
			`Managed object for overall server process.`

help	Provides a summary of all the available methods for the Help object.	• Parameters: None • Returns: string	Example output: WASX7028I: The Help object has two purposes: First, provide general help information for the bjects supplied by the wsadmin tool for scripting: Help, AdminApp, AdminConfig, and AdminControl. Second, provide a means to obtain interface information about the MBeans that run in the system. For this purpose, a variety of commands are available to get information about the operations,attributes, and other interface information about particular MBeans. The following commands are supported by Help; more detailed information about each of these commands is available by using the "help" command of Help and by supplying the name of the command as an argument. attributes given an MBean, returns help for attributes operations given an MBean, returns help for operations constructors given an MBean, returns help for constructors description given an MBean, returns help for description notifications given an MBean, returns help for notifications classname given an MBean, returns help for class name all given an MBean, returns help for all the previous help returns this help text AdminControl returns general help text for the AdminControl object AdminConfig returns general help text for the AdminConfig object AdminApp returns general help text for the AdminApp object AdminTask returns general help text for the AdminTask object wsadmin returns general help text for the wsadmin script launcher message given a message ID, returns an explanation and a user action

message	Displays information for a message ID.	• Parameters: message ID
		• Returns: string
			Example usage:
			Using Jacl:
			`$Help message CNTR0005W`
			Using Jython:
			`print Help.message('CNTR0005W')`
			Example output:
			`Explanation: The container was unable to passivate an enterprise bean due to exception {2}User action: Take action based upon message in exception {2}`

notifications	Provides a summary of all the notifications that the MBean defines by name.	• Parameters: name - string
		• Returns: string
			Example usage:
			Using Jacl:
			`$Help notifications [$AdminControl queryNames type=TraceService,process=server1, node=pongo,*]`
			Using Jython:
			`print Help.notifications (AdminControl.queryNames('type=TraceService, process=server1,node=pongo,*'))`
			Example output:
			`Notification`
			`websphere.messageEvent.audit`
			`websphere.messageEvent.fatal`
			`websphere.messageEvent.error`
			`websphere.seriousEvent.info`
			`websphere.messageEvent.warning`
			`jmx.attribute.changed`

operations	Provides a summary of all the operations that the MBean defines by name.	• Parameters: name - string
		• Returns: string
			Example usage:
			Using Jacl:
			`$Help operations [$AdminControl queryNames`
			`type=TraceService,process=server1,node=pongo,*]`
			Using Jython:
			`print Help.operations(AdminControl.queryNames`
			`('type=TraceService,process=server1,node=pongo,*'))`
			Example output:
			`Operation`
			`int getRingBufferSize()`
			`void setRingBufferSize(int)`
			`java.lang.String getTraceSpecification()`
			`void setTraceState(java.lang.String)`
			`void appendTraceString(java.lang.String)`
			`void dumpRingBuffer(java.lang.String)`
			`void clearRingBuffer()`
			`[Ljava.lang.String;`
			`listAllRegisteredComponents()`
			`[Ljava.lang.String;`
			`listAllRegisteredGroups()`
			`[Ljava.lang.String;`
			`listComponentsInGroup(java.lang.String)`
			`[Lcom.ibm.websphere.ras.TraceElementState;`
			`getTracedComponents()`
			`[Lcom.ibm.websphere.ras.TraceElementState;`
			`getTracedGroups()`
			`java.lang.String`
			`   getTraceSpecification(java.lang.String)`
			`void processDumpString(java.lang.String)`
			`void checkTraceString(java.lang.String)`
			`void setTraceOutputToFile(java.lang.String,`
			`int, int, java.lang.String)`
			`void setTraceOutputToRingBuffer`
			`(int, java.lang.String)`
			`java.lang.String rolloverLogFileImmediate`
			`(java.lang.String, java.lang.String)`

operations	Provides the signature of the opname operation for the MBean that is defined by name.	• Parameters: name - string, opname - string
		• Returns: string
			Example usage:
			Using Jacl:
			`$Help operations [$AdminControl queryNames`
			`type=TraceService,process=server1,node=pongo,*]`
			` processDumpString`
			Using Jython:
			`print Help.operations(AdminControl.queryNames`
			`('type=TraceService,process=server1,node=pongo,*'),`
			`'processDumpString')`
			Example output:
			`void processDumpString(string)`
			
			`Description: Write the contents of the Ras`
			`services ring buffer to the specified file.`
			
			`Parameters:`
			
			`Type         string`
			`Name         dumpString`
			`Description  A String in the specified format`
			`to process or null.`

Commands for the AdminConfig object

Use the AdminConfig object to invoke configuration commands and to create or change elements of the WebSphere Application Server configuration, for example, creating a data source.

You can start the scripting client without a running server, if you only want to use local operations. To run in local mode, use the -conntype NONE option to start the scripting client. You receive a message that you are running in the local mode. If a server is currently running, running the AdminConfig tool in local mode is not recommended. This is because any configuration changes made in local mode will not be reflected in the running server configuration and vice versa. If you save a conflicting configuration, you could corrupt the configuration. In a deployment manager environment, configuration updates are available only if a scripting client is connected to a deployment manager. When connected to a node agent or a managed application server, you will not be able to update the configuration because the configuration for these server processes are copies of the master configuration which resides in the deployment manager. The copies are created on a node machine when a configuration synchronization occurs between the deployment manager and the node agent. Make configuration changes to the server processes by connecting a scripting client to a deployment manager. For this reason, to change a configuration, do not run a scripting client in local mode on a node machine. It is not a supported configuration.

The following commands are available for the AdminConfig object:

Command name:	**Description:**	**Parameters and return values:**	**Examples:**

attributes	Returns a list of the top level attributes for a given type.	• Parameters: object type
		The name of the object type that you input here is the one based on the XML configuration files and does not have to be the same name that the administrative console displays.
		• Returns: A list of attributes.
			Example usage:
			Using Jacl:
			`$AdminConfig attributes ApplicationServer`
			Using Jython:
			`print AdminConfig.attributes`
			` ('ApplicationServer')`
			Example output:
			`"properties Property*" "serverSecurity`
			` ServerSecurity"`
			`"server Server@" "id Long" "stateManagement`
			` StateManageable"`
			`"name String" "moduleVisibility EEnumLiteral`
			` (MODULE,`
			`COMPATIBILITY, SERVER, APPLICATION)"`
			` "services Service*"`
			`"statisticsProvider StatisticsProvider"`

checkin	Checks a file that the document URI describes into the configuration repository.
	This method only applies to deployment manager configurations.
		• Parameters: document URI, filename, opaque object
		• Returns: None
			Example usage:
			Using Jacl:
			`$AdminConfig checkin`
			`cells/MyCell/Node/MyNode/serverindex.xml`
			`c:\\mydir\myfile $obj`
			Using Jython:
			`AdminConfig.checkin`
			`('cells/MyCell/Node/`
			`  MyNode/serverindex.xml',`
			`'c:\mydir\myfile',  obj)`
			The document URI is relative to the root of the configuration repository, for example, `c:\WebSphere\AppServer\config`.
			The file that is specified by `filename` is used as the source of the file to check. The *opaque* object is an object that the **extract** command of the AdminConfig object returns by a prior call.

convertToCluster	Converts a server so that it is the first member of a new server cluster.	• Parameters: server ID, cluster name
		• Returns: The configuration ID of the new cluster.
			Example usage:
			Using Jacl:
			`set serverid [$AdminConfig getid`
			`  /Server:myServer/]`
			`$AdminConfig convertToCluster`
			`  $serverid myCluster`
			Using Jython:
			`serverid = AdminConfig.getid`
			`  ('/Server:myServer/')`
			`AdminConfig.convertToCluster`
			`  (serverid, 'myCluster')`
			Example output:
			`myCluster`
			`(cells/mycell/clusters/myCluster|`
			`  cluster.xml#ClusterMember_2`

create	Creates configuration objects.	- Parameters using Jacl: type- string; parent ID- string; attributes- string
		- Parameters using Jython: type- string; parent ID- string; attributes- string or type- string; parent ID- string; attributes- Jython list
		- Returns: A string with configuration object names.
			The name of the object type that you input here is the one that is based on the XML configuration files. This name does not have to be the same name that the administrative console displays.
			Example usage:
			Using Jacl:
			`set jdbc1 [$AdminConfig getid`
			`  /JDBCProvider:jdbc1/]`
			`$AdminConfig create DataSource $jdbc1`
			`  {{name ds1}}`
			Using Jython with string attributes:
			`jdbc1 = AdminConfig.getid`
			`  ('/JDBCProvider:jdbc1/')`
			`AdminConfig.create('DataSource', jdbc1,`
			`  '[[name ds1]]')`
			Using Jython with object attributes:
			`jdbc1 = AdminConfig.getid`
			`  ('/JDBCProvider:jdbc1/')`
			`AdminConfig.create('DataSource', jdbc1,`
			`  [['name', 'ds1']])`
			Example output:
			`ds1(cells/mycell/nodes/DefaultNode/servers/server1|`
			`resources.xml#DataSource_6)`

createClusterMember	Creates a new server as a member of an existing cluster.
	This method creates a new server object on the node that the node id parameter specifies. This server is created as a new member of the existing cluster that is specified by the cluster id parameter, and contains attributes that are specified in the member attributes parameter. The server is created using the server template that is specified by the template id attribute, and that contains the name specified by the memberName attribute. The memberName attribute is required.
		• Parameters using Jacl: cluster ID- string; node ID- string; member attributes- string
		• Parameters using Jython: cluster ID- string; node ID- string; member attributes- string or cluster ID- string; node ID- string; member attributes- Jython list
		• Returns: The configuration ID of the new cluster member.
			The name of the object type that you input here is the one that is based on the XML configuration files. This name does not have to be the same name that the administrative console displays.
			Example usage:
			Using Jacl:
			`set clid [$AdminConfig getid`
			`/ServerCluster:myCluster/]`
			`set nodeid [$AdminConfig getid /Node:mynode/]`
			`$AdminConfig createClusterMember $clid`
			`$nodeid`
			`{{memberName newMem1} {weight 5}}`
			Using Jython with string attributes:
			`clid = AdminConfig.getid`
			`('/ServerCluster:myCluster/')`
			`nodeid = AdminConfig.getid('/Node:mynode/')`
			`AdminConfig.createClusterMember`
			`(clid, nodeid,`
			`'[[memberName newMem1] [weight 5]]')`
			Using Jython with object attributes:
			`clid = AdminConfig.getid('/ServerCluster:myCluster/')`
			`nodeid = AdminConfig.getid('/Node:mynode/')`
			`AdminConfig.createClusterMember(clid, nodeid,`
			`[['memberName', 'newMem1'], ['weight', 5]])`
			Example output:
			`myCluster`
			`(cells/mycell/clusters/myCluster|cluster.xml#`
			`ClusterMember_2)`

createDocument	Creates a new document in the configuration repository.
	The documentURI parameter names the document to create in the repository. The filename parameter must be a valid local file name where the contents of the document exist.
		• Parameters: documentURI, filename
		• Returns: None
			Example usage:
			Using Jacl:
			`$AdminConfig createDocument cells/mycell/myfile.xml`
			`c:\\mydir\\myfile`
			Using Jython:
			`AdminConfig.createDocument('cells/mycell/myfile.xml',`
			`'c:\mydir\myfile')`

createUsing Template	Creates a type of object with the given parent, using a template.	• Parameters using Jacl: type-string; parent id-string; attributes-string; template ID-string
		• Parameters using Jython: type-string; parent id- string; attributes-string; template ID-string or type-string; parent id-string; attributes-Jython list; template ID-string
		• Returns: The configuration ID of a new object.
			Example usage:
			Using Jacl:
			`set node [$AdminConfig getid /Node:`*mynode*`/]`
			`set templ [$AdminConfig listTemplates JDBCProvider`
			`"DB2 JDBC Provider (XA)"]`
			`$AdminConfig createUsingTemplate JDBCProvider $node`
			`{{name `*newdriver*`}} $templ`
			Using Jython using string attributes:
			`node = AdminConfig.getid('/Node:mynode/')`
			`templ = AdminConfig.listTemplates('JDBCProvider',`
			`"DB2 JDBC Provider (XA)")`
			`AdminConfig.createUsingTemplate('JDBCProvider',`
			` node, '[[name newdriver]]', templ)`
			Using Jython using object attributes:
			`node = AdminConfig.getid('/Node:mynode/')`
			`templ = AdminConfig.listTemplates('JDBCProvider',`
			`"DB2 JDBC Provider (XA)")`
			`AdminConfig.createUsingTemplate('JDBCProvider',`
			`node, [['name', 'newdriver']], templ)`

defaults	Displays the default values for attributes of a given type.
	This method displays all of the possible attributes contained by an object of a specific type. If the attribute has a default value, this method also displays the type and default value for each attribute.
		• Parameters: type
		The name of the object type that you input here is the one based on the XML configuration files. This name does not have to be the same name that the administrative console displays.
		• Returns: A string that contains a list of attributes with its type and value.
			Example usage:
			Using Jacl:
			`$AdminConfig defaults TuningParams`
			Using Jython:
			`print AdminConfig.defaults('TuningParams')`
			Example output:
			`Attribute                Type     Default`
			
			`usingMultiRowSchema      Boolean  false`
			`maxInMemorySessionCount  Integer  1000`
			`allowOverflow            Boolean  true`
			`scheduleInvalidation     Boolean  false`
			`writeFrequency           ENUM`
			`writeInterval            Integer  120`
			`writeContents            ENUM`
			`invalidationTimeout      Integer  30`
			`invalidationSchedule     InvalidationSchedule`

deleteDocument	Deletes a document from the configuration repository.
	The documentURI parameter names the document to delete from the repository.
		• Parameters: documentURI
		• Returns: None
			Example usage:
			Using Jacl:
			`$AdminConfig deleteDocument cells/mycell/myfile.xml`
			Using Jython:
			`AdminConfig.deleteDocument('cells/mycell/myfile.xml')`

existsDocument	Tests for the existence of a document in the configuration repository.
	The documentURI parameter names the document to test in the repository.
		• Parameters: documentURI
		• Returns: A true value, if the document exists.
			Example usage:
			Using Jacl:
			`$AdminConfig existsDocument cells/mycell/myfile.xml`
			Using Jython:
			`AdminConfig.existsDocument('cells/mycell/myfile.xml')`
			Example output:
			`1`

extract	Extracts a configuration repository file that is described by the document URI and places it in the file named by `filename`. This method only applies to deployment manager configurations.	• Parameters: document URI, filename
		• Returns: An opaue java.lang.Object to use when checking in the file.
			Example usage:
			Using Jacl:
			`set obj [$AdminConfig extract cells/MyCell/nodes/`
			`MyNode/serverindex.xml`
			`c:\\mydir\myfile]`
			Using Jython:
			`obj = AdminConfig.extract('cells/MyCell/nodes/`
			`MyNode/serverindex.xml',`
			`'c:\mydir\myfile')`
			The document URI is relative to the root of the configuration repository, for example, `c:\WebSphere\AppServer\config`.
			If the file that is specified by the filename parameter exists, the extracted file replaces it.

getCrossDocumentValidationEnabled	Returns a message with the current cross-document enablement setting.
	This method returns true if cross-document validation is enabled.
		• Parameters: None
		• Returns: A string that contains the message with the cross-document validation setting.
			Example usage:
			Using Jacl:
			`$AdminConfig getCrossDocumentValidationEnabled`
			Using Jython:
			`print AdminConfig.getCrossDocumentValidationEnabled()`
			Example output:
			`WASX7188I: Cross-document validation enablement set to`
			`true`

getid	Returns the configuration ID of an object.	• Parameters: containment path
		• Returns: The configuration ID for an object that is described by the containment path.
			Example usage:
			Using Jacl:
			`$AdminConfig getid /Cell:testcell/Node:testNode/`
			`JDBCProvider:Db2JdbcDriver/`
			Using Jython:
			`AdminConfig.getid('/Cell:testcell/Node:testNode/`
			`JDBCProvider:Db2JdbcDriver/')`
			Example output:
			`Db2JdbcDriver(cells/testcell/nodes/testnode|`
			`resources.xml#JDBCProvider_1)`

getObjectName	Returns a string version of the object name for the corresponding running MBean.
	This method returns an empty string if no corresponding running MBean exists.
		- Parameters: configuration ID
		- Returns: A string that contains the object name.
			Example usage:
			Using Jacl:
			`set server [$AdminConfig getid /Node:mynode/Server:server1/]`
			`$AdminConfig getObjectName $server`
			Using Jython:
			`server = AdminConfig.getid('/Node:mynode/Server:server1/')`
			`AdminConfig.getObjectName(server)`
			Example output:
			`WebSphere:cell=mycell,name=server1,mbeanIdentifier=cells/`
			`mycell/nodes/mynode/servers/server1/server.xml#Server_1,`
			`type=Server,node=mynode,process=server1,processType=`
			`UnManagedProcess`

getSaveMode	Returns the mode that is used when you invoke a **save** command.
	Possible values include the following:
	- overwriteOnConflict - Saves changes even if they conflict with other configuration changes
	- rollbackOnConflict - Causes a save operation to fail if changes conflict with other configuration changes. This value is the default.
		- Parameters: None
		- Returns: A string that contains the current save mode setting.
			Example usage:
			Using Jacl:
			`$AdminConfig getSaveMode`
			Using Jython:
			`print AdminConfig.getSaveMode()`
			Example output:
			`rollbackOnConflict`

getValidationLevel	Returns the validation used when files are extracted from the repository.	• Parameters: None
		• Returns: A string that contains the validation level.
			Example usage:
			Using Jacl:
			`$AdminConfig getValidationLevel`
			Using Jython:
			`AdminConfig.getValidationLevel()`
			Example output:
			`WASX7189I: Validation level set to HIGH`

getValidation SeverityResult	Returns the number of validation messages with the given severity from the most recent validation.	• Parameters: severity
		• Returns: A string that indicates the number of validation messages of the given severity.
			Example usage:
			Using Jacl:
			`$AdminConfig getValidationSeverityResult 1`
			Using Jython:
			`AdminConfig.getValidationSeverityResult(1)`
			Example output:
			`16`

hasChanges	Returns `true` if unsaved configuration changes exist.	• Parameters: None
		• Returns: A string that indicates whether unsaved configuration changes exist.
			Example usage:
			Using Jacl:
			`$AdminConfig hasChanges`
			Using Jython:
			`AdminConfig.hasChanges()`
			Example output:
			`1`

help	Displays static help information for the AdminConfig object.	• Parameters: None
		• Returns: A list of options.
			Example usage:
			Using Jacl:
			`$AdminConfig help`
			Using Jython:
			`print AdminConfig.help()`
			Example output:
			`WASX7053I: The AdminConfig object communicates with the`
			`configuration service in a WebSphere Application Server`
			`to manipulate configuration data`
			`for an Application Server installation.  The AdminConfig`
			`object has commands to list, create,`
			`remove, display, and modify configuration data, as well`
			`as commands to display information about configuration`
			`data types.`
			
			`Most of the commands supported by the AdminConfig object`
			`operate in two modes:`
			`the default mode is one in which the AdminConfig object`
			`communicates with the Application Server to accomplish`
			` its tasks.  A local mode is also possible, in which no`
			`server communication takes place.  The local mode of`
			`operation is invoked by bringing up the scripting client`
			`without a server connected using the command line`
			`"-conntype NONE" option or setting the`
			`"com.ibm.ws.scripting.connectionType=NONE" property in`
			`the wsadmin.properties file.`
			
			`The following commands are supported by the AdminConfig`
			`object; more detailed information about each of these`
			`commands is available by using the help command of the`
			`AdminConfig object and by supplying the name of the`
			`command as an argument.`
			
			`attributes      Shows the attributes for a given type`
			`checkin         Checks a file into the configuration`
			`          repository.`
			`convertToCluster`
			`                Converts a server to be the first member`
			`         of a new server cluster`
			`create          Creates a configuration object, given`
			`         a type, a parent, and a list of attributes,`
			`         and optionally an attribute name for the`
			`                new object`
			`createClusterMember`
			`                Creates a new server that is a member of an`
			`                existing cluster.`
			`createDocument  Creates a new document in the configuration`
			`         repository.`
			`installResourceAdapter`
			`                Installs a J2C resource adapter with the`
			`         given RAR file name and an option string`
			`         in the node.`
			`createUsingTemplate`
			`            Creates an object using a particular`
			`         template type.`
			`defaults      Displays the default values for the`
			`         attributes of a given type.`
			`deleteDocument  Deletes a document from the configuration`
			`         repository.`
			`existsDocument  Tests for the existence of a document in`
			`         the configuration repository.`

help continued			extract Extracts a file from the configuration repository. getCrossDocumentValidationEnabled Returns true if cross-document validation is enabled. getid Show the configuration ID of an object, given a string version of its containment getObjectName Given a configuration ID, returns a string version of the ObjectName for the corresponding running MBean, if any. getSaveMode Returns the mode used when "save" is invoked getValidationLevel Returns the validation that is used when files are extracted from the repository. getValidationSeverityResult Returns the number of messages of a given severity from the most recent validation. hasChanges Returns true if unsaved configuration changes exist help Shows help information list Lists all the configuration objects of a given type listTemplates Lists all the available configuration templates of a given type. modify Changes the specified attributes of a given configuration object parents Shows the objects which contain a given type queryChanges Returns a list of unsaved files remove Removes the specified configuration object required Displays the required attributes of a given type. reset Discards the unsaved configuration changes save Commits the unsaved changes to the configuration repository setCrossDocumentValidationEnabled Sets the cross-document validation enabled mode. setSaveMode Changes the mode used when "save" is invoked setValidationLevel Sets the validation used when files are extracted from the repository. show Shows the attributes of a given configuration object showall Recursively shows the attributes of a given configuration object, and all the objects that are contained within each attribute. showAttribute Displays only the value for the single attribute that is specified. types Shows the possible types for configuration validate Invokes validation

installResource Adapter	Installs a Java 2 Connector (J2C) resource adapter with the given Resource Adapter Archive (RAR) file name and an option string in the node.
	The RAR file name is the fully qualified file name that resides in the node that you specify. The valid options include the following options:
	• `rar.name`
	• `rar.desc`
	• `rar.archivePath`
	• `rar.classpath`
	• `rar.nativePath`
	• `rar.threadPoolAlias`
	• `rar.propertiesSet`
		• Parameters: RAR file name, node, options
		• Returns: The configuration ID of the new J2CResourceAdapter object.
			Example usage:
			Using Jacl:
			`$AdminConfig installResourceAdapter c:/rar/mine.rar mynode {-rar.name myResourceAdapter -rar.desc "My rar file"}`
			Using Jython:
			`print AdminConfig.installResourceAdapter('c:/rar/mine.rar', 'mynode', '[-rar.name myResourceAdapter -rar.desc "My rar file"]')`
			Example output:
			`myResourceAdapter(cells/mycell/nodes/mynode|resources.xml#J2CResourceAdapter_1)`

installResource Adapter continued	The rar.name option is the name for the J2C resource adapter. If you do not specify this option, the display name in the RAR deployment descriptor is used. If that name is not specified, the RAR file name is used. The rar.desc option is a description of the J2CResourceAdapter. The rar.archivePath is the name of the path where you extract the file. If you do not specify this option, the archive is extracted to the $\{CONNECTOR_INSTALL_ROOT\} directory. The rar.classpath option is the additional class path. rar.propertiesSet is constructed with the following: name String value String type String *desc String *required true/false * means the item is optional Each attribute of the property are specified in a set of {}. A property is specified in a set of {}. You can specify multiple properties in {}.		

installResource Adapter continued	When you edit the installed application with the embedded RAR, only existing J2C connection factory, J2C activation specs, and J2C administrative objects will be edited. No new J2C objects will be created.		

list	Returns a list of objects of a given type, possibly scoped by a parent.	- Parameters: Object type
		The name of the object type that you input here is the one that is based on the XML configuration files and does not have to be the same name that the administrative console displays.
		- Returns: A list of objects.
			Example usage:
			Using Jacl:
			`$AdminConfig list JDBCProvider`
			Using Jython:
			`print AdminConfig.list('JDBCProvider')`
			Example output:
			`Db2JdbcDriver(cells/mycell/nodes/DefaultNode|`
			`  resources.xml#JDBCProvider_1)`
			`Db2JdbcDriver(cells/mycell/nodes/DefaultNode/servers/`
			`  deploymentmgr|`
			`resources.xml#JDBCProvider_1)`
			`Db2JdbcDriver(cells/mycell/nodes/DefaultNode/servers/`
			`nodeAgent|resources.xml#JDBCProvider_1)`

listTemplates	Displays a list of template object IDs.	- Parameters: object type
		The name of the object type that you input here is the one that is based on the XML configuration files and does not have to be the same name that the administrative console displays.
		- Returns: A list of template IDs.
			Example usage:
			Using Jacl:
			`$AdminConfig listTemplates JDBCProvider`
			Using Jython:
			`print AdminConfig.listTemplates('JDBCProvider')`
			This example displays a list of all the JDBCProvider templates that are available on the system.

<table>
<tr><td>modify</td><td>Supports the modification of object attributes.</td><td>• Parameters using Jacl: object-string; attributes-string
• Parameters using Jython: object-string; attributes-string or object-string; attributes-Jython list
• Returns: None</td><td>Example usage:
Using Jacl:
$AdminConfig modify ConnFactory1(cells/mycell/nodes/ DefaultNode/servers/ deploymentmgr|resources.xml#GenericJMSConnectionFactory_1) {{userID newID} {password newPW}}
Using Jython with string attributes:
AdminConfig.modify('ConnFactory1(cells/mycell/nodes/ DefaultNode/servers/ deploymentmgr|resources.xml#GenericJMSConnectionFactory_1)', '[[userID newID] [password newPW]]')
Using Jython with object attributes:
AdminConfig.modify('ConnFactory1(cells/mycell/nodes/ DefaultNode/servers/ deploymentmgr|resources.xml#GenericJMSConnectionFactory_1)', [['userID', 'newID'], ['password', 'newPW']])</td></tr>
<tr><td>parents</td><td>Obtains information about object types.</td><td>• Parameters: object type
The name of the object type that you input here is the one that is based on the XML configuration files and does not have to be the same name that the administrative console displays.
• Returns: A list of object types.</td><td>Example usage:
Using Jacl:
$AdminConfig parents JDBCProvider
Using Jython:
AdminConfig.parents('JDBCProvider')
Example output:
Cell
Node
Server</td></tr>
<tr><td>queryChanges</td><td>Returns a list of unsaved configuration files.</td><td>• Parameters: None
• Returns: A string that contains a list of files with unsaved changes.</td><td>Example usage:
Using Jacl:
$AdminConfig queryChanges
Using Jython:
AdminConfig.queryChanges()
Example output:
WASX7146I: The following configuration files contain unsaved changes:
cells/mycell/nodes/mynode/servers/server1|resources.xml</td></tr>
<tr><td>remove</td><td>Removes a configuration object.</td><td>• Parameters: Object
• Returns: None</td><td>Example usage:
Using Jacl:
$AdminConfig remove ds1(cells/mycell/nodes/DefaultNode/ servers/server1:resources.xml#DataSource_6)
Using Jython:
AdminConfig.remove('ds1(cells/mycell/nodes/DefaultNode/ servers/server1:resources.xml#DataSource_6)')</td></tr>
</table>

required	Displays the required attributes that are contained by an object of a certain type.	• Parameters: Type
		The name of the object type that you input here is the one that is based on the XML configuration files. It does not have to be the same name that the administrative console displays.
		• Returns: A string that contains a list of the required attributes with its type.
			Example usage:
			Using Jacl:
			`$AdminConfig required URLProvider`
			Using Jython:
			`print AdminConfig.required('URLProvider')`
			Example output:
			`Attribute                        Type`
			`streamHandlerClassName           String`
			`protocol                         String`

reset	Resets the temporary workspace that holds updates to the configuration.	• Parameters: None
		• Returns: None
			Example usage:
			Using Jacl:
			`$AdminConfig reset`
			Using Jython:
			`AdminConfig.reset()`

save	Saves changes in the configuration repository.	• Parameters: None
		• Returns: None
			Example usage:
			Using Jacl:
			`$AdminConfig save`
			Using Jython:
			`AdminConfig.save()`

setCrossDocumentValidationEnabled	Sets the cross-document validation enabled mode. Values include `true` or `false`.	• Parameters: Flag
		• Returns: None
			Example usage:
			Using Jacl:
			`$AdminConfig setCrossDocumentValidationEnabled true`
			Using Jython:
			`AdminConfig.setCrossDocumentValidationEnabled('true')`

setSaveMode	Toggles the behavior of the **save** command. The default value is `rollbackOnConflict`. When a conflict is discovered while saving, the unsaved changes are not committed. The alternative value is `overwriteOnConflict`, which saves the changes to the configuration repository even if conflicts exist.	• Parameters: Mode
		• Returns: None
			Example usage:
			Using Jacl:
			`$AdminConfig setSaveMode overwriteOnConflict`
			Using Jython:
			`AdminConfig.setSaveMode('overwriteOnConflict')`

setValidationLevel	Sets the validation that is used when files are extracted from the repository.
	Five validation levels are available: `none`, `low`, `medium`, `high`, or `highest`.
		• Parameters: Level
		• Returns: A string that contains the validation level setting.
			Example usage:
			Using Jacl:
			`$AdminConfig setValidationLevel high`
			Using Jython:
			`AdminConfig.setValidationLevel('high')`
			Example output:
			`WASX7189I: Validation level set to HIGH`

show	Returns the top-level attributes of the given object.	• Parameters: Object, attributes
		• Returns: A string that contains the attribute value.
			Example usage:
			Using Jacl:
			`$AdminConfig show Db2JdbcDriver(cells/mycell/ nodes/DefaultNode|resources.xm#JDBCProvider_1)`
			Example output with Jacl:
			`{name "Sample Datasource"} {description "Data source for the Sample entity beans"}`
			Using Jython:
			`print AdminConfig.show('Db2JdbcDriver(cells/mycell/nodes/ DefaultNode|resources.xm#JDBCProvider_1)')`
			Example output with Jython:
			`[name "Sample Datasource"] [description "Data source for the Sample entity beans"]`

showall	Recursively shows the attributes of a given configuration object.	- Parameters: Object, attributes
		- Returns: A string that contains the attribute value.
			Example usage:
			
			Using Jacl:
			
			$AdminConfig showall "Default Datasource(cells/mycell/
			nodes/DefaultNode/servers/server1:resources.xml#
			DataSource_1)
			
			Example output with Jacl:
			
			{authMechanismPreference BASIC_PASSWORD}
			{category default}
			{connectionPool {{agedTimeout 0}
			{connectionTimeout 1000}
			{maxConnections 30}
			{minConnections 1}
			{purgePolicy FailingConnectionOnly}
			{reapTime 180}
			{unusedTimeout 1800}}}
			{datasourceHelperClassname
			  com.ibm.websphere.rsadapter.CloudscapeDataStoreHelper}
			{description "Datasource for the WebSphere Default
			  Application"}
			{jndiName DefaultDatasource}
			{name "Default Datasource"}
			{propertySet {{resourceProperties {{{description
			"Location of Cloudscape default database."}
			{name databaseName}
			{type string}
			{value ${WAS_INSTALL_ROOT}/bin/DefaultDB}}
			  {{name remoteDataSourceProtocol}
			{type string}
			{value {}}} {{name shutdownDatabase}
			{type string}
			{value {}}} {{name dataSourceName}
			{type string}
			{value {}}} {{name description}
			{type string}
			{value {}}} {{name connectionAttributes}
			{type string}
			{value {}}} {{name createDatabase}
			{type string}
			{value {}}}}}}}
			{provider "Cloudscape JDBC Driver(cells/pongo/nodes/pongo/
			servers/server1|resources.xml#JDBCProvider_1)"}
			{relationalResourceAdapter "WebSphere Relational
			  Resource Adapter(cells/pongo/
			nodes/pongo/servers/server1|resources.xml#builtin_rra)"}
			{statementCacheSize 0}
			
			Using Jython:
			
			AdminConfig.showall("Default Datasource(cells/mycell/nodes/
			DefaultNode/servers/server1:resources.xml#DataSource_1)")
			
			Example output with Jython:
			
			 [authMechanismPreference BASIC_PASSWORD]
			[category default]
			[connectionPool [[agedTimeout []]
			[connectionTimeout 1000]
			[maxConnections 30]
			[minConnections 1]
			[purgePolicy FailingConnectionOnly]
			[reapTime 180]
			[unusedTimeout 1800]]]
			[datasourceHelperClassname
			 com.ibm.websphere.rsadapter.CloudscapeDataStoreHelper]
			[description "Datasource for the WebSphere Default
			 Application"]
			[jndiName DefaultDatasource]
			[name "Default Datasource"]

showall continued			[propertySet [[resourceProperties [[[description "Location of Cloudscape default database."] [name databaseName] [type string] [value ${WAS_INSTALL_ROOT}/bin/DefaultDB]] [[name remoteDataSourceProtocol] [type string] [value []]] [[name shutdownDatabase] [type string] [value []]] [[name dataSourceName] [type string] [value []]] [[name description] [type string] [value []]] [[name connectionAttributes] [type string] [value []]] [[name createDatabase] [type string] [value []]]]]]] [provider "Cloudscape JDBC Driver(cells/pongo/nodes/pongo/ servers/server1|resources.xml#JDBCProvider_1)"] [relationalResourceAdapter "WebSphere Relational Resource Adapter(cells/pongo/nodes/pongo/servers/server1| resources.xml#builtin_rra)"] [statementCacheSize 0]

showAttribute	Displays only the value for the single attribute that you specify.
	The output of this command is different from the output of the **show** command when a single attribute is specified. The **showAttribute** command does not display a list that contains the attribute name and value. It only displays the attribute value.
		• Parameters: Configuration ID, attribute
		• Returns: A string that contains the attribute value.
			Example usage:
			Using Jacl:
			`set ns [$AdminConfig getid /Node:mynode/]`
			`$AdminConfig showAttribute $ns hostName`
			Using Jython:
			`ns = AdminConfig.getid('/Node:mynode/')`
			`print AdminConfig.showAttribute(ns, 'hostName')`
			Example output:
			`mynode`

types	Returns a list of the configuration object types that you can manipulate.	• Parameters: None
		• Returns: A list of object types.
			Example usage:
			Using Jacl:
			`$AdminConfig types`
			Using Jython:
			`print AdminConfig.types()`
			Example output:
			`AdminService`
			`Agent`
			`ApplicationConfig`
			`ApplicationDeployment`
			`ApplicationServer`
			`AuthMechanism`
			`AuthenticationTarget`
			`AuthorizationConfig`
			`AuthorizationProvider`
			`AuthorizationTableImpl`
			`BackupCluster`
			`CMPConnectionFactory`
			`CORBAObjectNameSpaceBinding`
			`Cell`
			`CellManager`
			`Classloader`
			`ClusterMember`
			`ClusteredTarget`
			`CommonSecureInteropComponent`

uninstallResourceAdapter	Uninstalls a Java 2 Connector (J2C) resource adapter with the given J2C resource adapter configuration ID and an option list.
	One option is valid for this command: * `force`
	This option forces the uninstallation of the resource adapter without checking whether the resource adapter is being used by an application. The application that is using it will not be uninstalled. If you do not specify the force option and the specified resource adapter is still in use, the resource adapter is not uninstalled.
		• Parameters: J2C resource adapter configuration ID, list of options
		• Returns: The configuration ID of J2CResourceAdapter object that is removed.
			Example usage:
			Using Jacl:
			`set j2cra [$AdminConfig getid /J2CResourceAdapter:MyJ2CRA/]`
			`$AdminConfig uninstallResourceAdapter $j2cra {-force}`
			`$AdminConfig save`
			Using Jython:
			`j2cra = AdminConfig.getid('/J2CResourceAdapter:MyJ2CRA/')`
			`print AdminConfig.uninstallResourceAdapter(j2cra, '[-force]')`
			`AdminConfig.save()`
			Example output:
			`WASX7397I: The following J2CResourceAdapter objects are removed:`
			`MyJ2CRA(cells/juniarti/nodes/juniarti|`
			`resources.xml#J2CResourceAdapter_1069433028609)`

uninstallResource Adapter continued	When you remove a J2CResourceAdapter object from the configuration repository, the installed directory will be removed at the time of synchronization. A stop request will be sent to the J2CResourceAdapter MBean that was removed.		

validate	Invokes validation.
	This command requests configuration validation results based on the files in your workspace, the value of the cross-document validation enabled flag, and the validation level setting. Optionally, you can specify a configuration ID to set the scope. If you specify a configuration ID, the scope of this request is the object named by the config id parameter.
		• Parameters: config id (optional)
		• Returns: A string that contains results of the validation.
			Example usage:
			Using Jacl:
			`$AdminConfig validate`
			Using Jython:
			`print AdminConfig.validate()`
			Example output:
			`WASX7193I: Validation results are logged in c:\WebSphere5\ AppServer\logs\wsadmin.valout: Total number of messages: 16 WASX7194I: Number of messages of severity 1: 16`

Commands for the AdminControl object

Use the AdminControl object to invoke operational commands that deal with running objects in the WebSphere Application Server. Many of the AdminControl commands have multiple signatures so that they can either invoke in a raw mode using parameters that are specified by Java Management Extensions (JMX), or by using strings for parameters. In addition to operational commands, the AdminControl object supports some utility commands for tracing, reconnecting with a server, and converting data types.

The following commands are available for the AdminControl object:

Command name:	Description:	Parameters and return values:	Examples:

completeObjectName	Creates a string representation of a complete ObjectName value that is based on a fragment. This command does not communicate with the server to find a matching ObjectName value. If it finds several MBeans that match the fragment, the command returns the first one.	• Parameters: name-java.lang.String
		• Returns: java.lang.String
			Example usage:
			Using Jacl:
			`set serverON [$AdminControl completeObjectName node=mynode,type=Server,*]`
			Using Jython:
			`serverON = AdminControl.completeObjectName ('node=mynode,type=Server,*')`

getAttribute	Returns the value of the attribute for the name that you provide.	• Parameters: name-java.lang.String; attribute-java.lang.String
		• Returns: java.lang.String
			Example usage:
			Using Jacl:
			`set objNameString [$AdminControl completeObjectName WebSphere:type=Server,*]`
			`$AdminControl getAttribute $objNameString processType`
			Using Jython:
			`objNameString = AdminControl.completeObjectName ('WebSphere:type=Server,*')`
			`AdminControl.getAttribute(objNameString, 'processType')`

getAttribute_jmx	Returns the value of the attribute for the name that you provide.	• Parameters: name-ObjectName; attribute-java.lang.String
		• Returns: java.lang.String
			Example usage:
			Using Jacl:
			`set objNameString [$AdminControl`
			` completeObjectName WebSphere:type=Server,*]`
			`set objName [java::new`
			`  javax.management.ObjectName`
			` $objNameString]`
			`$AdminControl getAttribute_jmx`
			` $objName processType`
			Using Jython:
			`objNameString =`
			`  AdminControl.completeObjectName`
			`('WebSphere:type=Server,*')`
			`import  javax.management  as  mgmt`
			`objName =  mgmt.ObjectName(objNameString)`
			`AdminControl.getAttribute_jmx(objName,`
			`'processType')`

getAttributes	Returns the attribute values for the names that you provide.	• Parameters using Jacl: name-String; attributes-java.lang.String
		• Parameters using Jython: name-String; attributes-java.lang.String or name-String; attributes-java.lang.Object[]
		• Returns: java.lang.String
			Example usage:
			Using Jacl:
			`set objNameString [$AdminControl`
			` completeObjectName WebSphere:type=Server,*]`
			`$AdminControl getAttributes $objNameString`
			` "cellName nodeName"`
			Using Jython with string attributes:
			`objNameString =`
			`  AdminControl.completeObjectname`
			` ('WebSphere:type=Server,*)`
			`AdminControl.getAttributes(objNameString,`
			` '[cellName nodeName]')`
			Using Jython with object attributes:
			`objNameString =`
			` AdminControl.completeObjectname`
			` ('WebSphere:type=Server,*)`
			`AdminControl.getAttributes(objNameString,`
			` ['cellName', 'nodeName'])`

getAttributes_jmx	Returns the attribute values for the names that you provide.	• Parameters: name-ObjectName; attributes-java.lang.String[]
		• Returns: javax.management.AttributeList
			Example usage:
			Using Jacl:
			`set objectNameString [$AdminControl`
			` completeObjectName WebSphere:type=Server,*]`
			`set objName [$AdminControl makeObjectName`
			` $objectNameString]`
			`set attrs [java::new {String[]} 2`
			` {cellName nodeName}]`
			`$AdminControl getAttributes_jmx $objName`
			` $attrs`
			Using Jython:
			`objectNameString = AdminControl.completeObjectName`
			` ('type=Server,*')`
			`objName = AdminControl.makeObjectName`
			` (objectNameString)`
			`attrs = ['cellName', 'nodeName']`
			`AdminControl.getAttributes_jmx(objName,`
			` attrs)`

getCell	Returns the name of the connected cell.	- Parameters: None
		- Returns: java.lang.String
			Example usage:
			Using Jacl:
			`$AdminControl getCell`
			Using Jython:
			`AdminControl.getCell()`
			Example output:
			`Mycell`

getConfigId	Creates a configuration ID from an ObjectName or an ObjectName fragment. Use this ID with the `$AdminConfig` command. Not all MBeans that run have configuration objects that correspond. If several MBeans correspond to an ObjectName fragment, a warning is created and a configuration ID builds for the first MBean it finds.	- Parameters: name-java.lang.String
		- Returns: java.lang.String
			Example usage:
			Using Jacl:
			`set threadpoolCID [$AdminControl getConfigId node=mynode,type=ThreadPool,*]`
			Using Jython:
			`threadpoolCID = AdminControl.getConfigId ('node=mynode,type=ThreadPool,*')`

getDefaultDomain	Returns the default domain name from the server.	- Parameters: None
		- Returns: java.lang.String
			Example usage:
			Using Jacl:
			`$AdminControl getDefaultDomain`
			Using Jython:
			`AdminControl.getDefaultDomain()`
			Example output:
			`WebSphere`

getDomainName	Returns the domain name from the server.	• Parameters: None
		• Returns: java.lang.String
			Example usage:
			Using Jacl:
			`$AdminControl getDomainName`
			Using Jython:
			`AdminControl.getDomainName()`
			Example output:
			`WebSphere`

getHost	Returns the name of your host.	• Parameters: None
		• Returns: java.lang.String
			Example usage:
			Using Jacl:
			`$AdminControl getHost`
			Using Jython:
			`AdminControl.getHost()`
			Example output:
			`myhost`

getMBeanCount	Returns the number of MBeans that are registered in the server.	• Parameters: None
		• Returns: java.lang.Integer
			Example usage:
			Using Jacl:
			`$AdminControl getMBeanCount`
			Using Jython:
			`AdminControl.getMBeanCount()`
			Example output:
			`114`

getMBeanInfo_jmx	Returns the Java Management Extension MBeanInfo structure that corresponds to an ObjectName value. No string signature exists for this command, because the Help object displays most of the information available from the **getMBeanInfo** command.	• Parameters: name-ObjectName
		• Returns: javax.management.MBeanInfo
			Example usage:
			Using Jacl:
			`set objectNameString [$AdminControl`
			` completeObjectName type=Server,*]`
			`set objName [$AdminControl makeObjectName`
			` $objectNameString]`
			`$AdminControl getMBeanInfo_jmx $objName`
			Using Jython:
			`objectNameString =`
			` AdminControl.completeObjectName('type=Server,*')`
			`objName = AdminControl.makeObjectName`
			` (objectNameString)`
			`AdminControl.getMBeanInfo_jmx(objName)`
			Example output:
			`javax.management.modelmbean.`
			` ModelMBeanInfoSupport@10dd5f35`

getNode	Returns the name of the connected node.	• Parameters: None
		• Returns: java.lang.String
			Example usage:
			Using Jacl:
			`$AdminControl getNode`
			Using Jython:
			`AdminControl.getNode()`
			Example output:
			`Myhost`

getPort	Returns the name of your port.	• Parameters: None
		• Returns: java.lang.String
			Example usage:
			Using Jacl:
			`$AdminControl getPort`
			Using Jython:
			`AdminControl.getPort()`
			Example output:
			`8877`

getPropertiesForDataSource	Deprecated, no replacement.
	This command incorrectly assumes the availability of a configuration service when running in connected mode.
		• Parameters: configId-java.lang.String
		• Returns: java.lang.String
			Example usage:
			Using Jacl:
			`set ds [lindex [$AdminConfig list DataSource] 0]`
			`$AdminControl getPropertiesForDataSource $ds`
			Using Jython:
			`ds = AdminConfig.list('DataSource')`
			
			`# get line separator`
			`import  java.lang.System  as  sys`
			`lineSeparator = sys.getProperty('line.separator')`
			
			`dsArray = ds.split(lineSeparator)`
			`AdminControl.getPropertiesForDataSource(dsArray[0])`
			Example output:
			`WASX7389E: Operation not supported -`
			` getPropertiesForDataSource command`
			`is not supported.`

getType	Returns the connection type.	• Parameters: None
		• Returns: java.lang.String
			Example usage:
			Using Jacl:
			`$AdminControl getType`
			Using Jython:
			`AdminControl.getType()`
			Example output:
			`SOAP`

help	Returns general help text for the AdminControl object.	• Parameters: None
		• Returns: java.lang.String
			Example usage:
			Using Jacl:
			`$AdminControl help`
			Using Jython:
			`AdminControl.help()`
			Example output:
			`WASX7027I: The AdminControl object enables`
			`the manipulation of MBeans that run in a`
			`WebSphere Application Server process.`
			`The number and type of MBeans that are`
			`available to the scripting client depend`
			`on the server to which the client is`
			`connected.  If the client is connected to a`
			`deployment manager, then all the MBeans`
			`running in the Deployment`
			`Manager are visible, as are all the MBeans`
			`running in the node agents that are`
			`connected to this deployment manager, and`
			`all the MBeans that run in the application`
			`servers on those nodes.`
			
			`The following commands are supported by`
			`the AdminControl object; more detailed`
			`information about each of these commands`
			`is available by using the "help" command`
			`of the AdminControl object and supplying`
			`the name of the command as an argument.`
			
			`Many of these commands support two`
			`different sets of signatures: one that`
			`accepts and returns strings, and one`
			`low-level set that works with JMX objects`
			`like ObjectName and AttributeList. In most`
			`situations, the string signatures are`
			`likely to be more useful,but JMX-object`
			`signature versions are supplied as well.`
			`Each of these JMX-object signature`
			`commands has "_jmx" appended to the`
			`command name,so an "invoke" command, as`
			`well as a "invoke_jmx" command are`
			`supported.`
			
			`completeObjectName`
			`  Return a String version`
			`  of an object name given a`
			`     template name`
			`getAttribute_jmx`
			`     Given ObjectName and name of`
			`   attribute, returns value of`
			`      attribute`
			`getAttribute`
			`  Given String version of ObjectName`
			`  and name of attribute,returns value`
			`  of attribute`
			`getAttributes_jmx`
			`     Given ObjectName and array of`
			`   attribute names, returns`
			`     AttributeList`
			`getAttributes`
			`  Given String version of ObjectName`
			`  and attribute names,returns String`
			`  of name value pairs`

help continued			getCell returns the cell name of the connected server getConfigId Given String version of ObjectName, return a config id for the corresponding configuration object, if any. getDefaultDomain returns "WebSphere" getDomainName returns "WebSphere" getHost returns String representation of connected host getMBeanCount returns number of registered beans getMBeanInfo_jmx Given ObjectName, returns MBeanInfo structure for MBean getNode returns the node name of the connected server getPort returns String representation of port in use getType returns String representation of connection type in use help Show help information invoke_jmx Given ObjectName, name of command, array of parameters and signature, invoke command on MBean specified invoke Invoke a command on the specified MBean isRegistered_jmx true if supplied ObjectName is registered isRegistered true if supplied String version of ObjectName is registered makeObjectName Return an ObjectName built with the given string queryNames_jmx Given ObjectName and QueryExp, retrieves set of ObjectNames that match. queryNames Given String version of ObjectName, retrieves String of ObjectNames that match. reconnect reconnects with server setAttribute_jmx Given ObjectName and Attribute object, set attribute for MBean specified setAttribute Given String version of ObjectName, attribute name and attribute value, set attribute for MBean specified setAttributes_jmx Given ObjectName and AttributeList object, set attributes for the MBean specified startServer Given the name of a server, start that server. stopServer Given the name of a server, stop that server. testConnection Test the connection to a DataSource object trace Set the wsadmin trace specification

help	Returns help text for the specific command of the AdminControl object. The command name is not case sensitive.	- Parameters: command-java.lang.String
		- Returns: java.lang.String
			Example usage:
			Using Jacl:
			`$AdminControl help getAttribute`
			Using Jython:
			`AdminControl.help('getAttribute')`
			Example output:
			`WASX7043I: command: getAttribute`
			`Arguments: object name, attribute`
			`Description: Returns value of "attribute"`
			`for the MBean described by "object name."`

invoke	Invokes the object operation without any parameter. Returns the result of the invocation.	- Parameters: name-java.lang.String; operationName-java.lang.String
		- Returns: java.lang.String
			Example usage:
			Using Jacl:
			`set objNameString [$AdminControl`
			` completeObjectName WebSphere:type=Server,*]`
			`$AdminControl invoke $objNameString stop`
			Using Jython:
			`objNameString =`
			` AdminControl.completeObjectName`
			` ('WebSphere:type=Server,*')`
			`AdminControl.invoke(objNameString, 'stop')`

invoke	Invokes the object operation using the parameter list that you supply. The signature generates automatically. The types of parameters are supplied by examining the MBeanInfo that the MBean supplies. Returns the string result of the invocation. The string that is returned is controlled by the Mbean method that you invoked. If the Mbean method is synchronous, then control is returned back to the wsadmin tool only when the operation is complete. If the Mbean method is asynchronous, control is returned back to the wsadmin tool immediately even though the invoked task might not be complete.	- Parameters: name-java.lang.String; operationName-java.lang.String; params-java.lang.String
		- Returns: java.lang.String
			Example usage:
			Using Jacl:
			`set objNameString [$AdminControl completeObjectName WebSphere:type=Server,*]`
			`$AdminControl invoke $objNameString appendTraceString com.ibm.*=all=enabled`
			Using Jython:
			`objNameString = AdminControl.completeObjectName ('WebSphere:type=Server,*')`
			`AdminControl.invoke(objNameString, 'appendTraceString', 'com.ibm.*=all=enabled')`

invoke	Invokes the object operation by conforming the parameter list to the signature. Returns the result of the invocation.	• Parameters: name-java.lang.String; operationName-java.lang.String; params-java.lang.String; sigs-java.lang.String
		• Returns: java.lang.String
			Example usage:
			Using Jacl:
			`set objNameString [$AdminControl`
			` completeObjectName WebSphere:type=Server,*]`
			`$AdminControl invoke $objNameString`
			` appendTraceString com.ibm.*=all=enabled`
			` java.lang.String`
			Using Jython:
			`objNameString = AdminControl.completeObjectName`
			` ('WebSphere:type=Server,*')`
			`AdminControl.invoke(objNameString,`
			` 'appendTraceString', 'com.ibm.*=all=enabled',`
			` 'java.lang.String')`

invoke_jmx	Invokes the object operation by conforming the parameter list to the signature. Returns the result of the invocation.	• Parameters: name-ObjectName; operationName-java.lang.String; params-java.lang.Object[]; signature-java.lang.String[]
		• Returns: java.lang.Object
			Example usage:
			`set objNameString [$AdminControl`
			` completeObjectName WebSphere:type=TraceService,*]`
			`set objName [java::new javax.management.ObjectName`
			` $objNameString]`
			`set parms [java::new {java.lang.Object[]}`
			` 1 com.ibm.ejs.sm.*=all=disabled]`
			`set signature [java::new`
			` {java.lang.String[]} 1 java.lang.String]`
			`$AdminControl invoke_jmx $objName`
			` appendTraceString $parms $signature`
			Using Jython:
			`objNameString = AdminControl.completeObjectName`
			`  ('WebSphere:type=TraceService,*')`
			`import  javax.management  as  mgmt`
			`objName =  mgmt.ObjectName(objNameString)`
			`parms = ['com.ibm.ejs.sm.*=all=disabled']`
			`signature = ['java.lang.String']`
			`AdminControl.invoke_jmx(objName,`
			` 'appendTraceString', parms, signature)`

isRegistered	If the ObjectName value is registered in the server, then the value is `true`.	• Parameters: name-java.lang.String
		• Returns: Boolean
			Example usage:
			Using Jacl:
			`set objNameString [$AdminControl`
			`completeObjectName WebSphere:type=Server,*]`
			`$AdminControl isRegistered $objNameString`
			Using Jython:
			`objNameString =`
			`AdminControl.completeObjectName`
			` ('WebSphere:type=Server,*')`
			`AdminControl.isRegistered(objNameString)`

isRegistered_jmx	If the ObjectName value is registered in the server, then the value is `true`.	• Parameters: name-ObjectName
		• Returns: Boolean
			Example usage:
			Using Jacl:
			`set objectNameString [$AdminControl completeObjectName type=Server,*]`
			`set objName [$AdminControl makeObjectName $objNameString]`
			`$AdminControl isRegistered_jmx $objName`
			Using Jython:
			`objectNameString = AdminControl.completeObjectName ('type=Server,*')`
			`objName = AdminControl.makeObjectName (objectNameString)`
			`AdminControl.isRegistered_jmx(objName)`

makeObjectName	A convenience command that creates an ObjectName value that is based on the strings input. This command does not communicate with the server, so the ObjectName value that results might not exist. If the string you supply contains an extra set of double quotes, they are removed. If the string does not begin with a Java Management Extensions (JMX) domain, or a string followed by a colon, then the WebSphere Application Server string appends to the name.	• Parameters: name-java.lang.String
		• Returns: javax.management.ObjectName
			Example usage:
			Using Jacl:
			`set objectNameString [$AdminControl completeObjectName type=Server,node=mynode,*]`
			`set objName [$AdminControl makeObjectName $ objNameString]`
			Using Jython:
			`objectNameString = AdminControl.completeObjectName ('type=Server,node=mynode,*')`
			`objName = AdminControl.makeObjectName (objectNameString)`

queryNames	Returns a string that lists all the ObjectName objects based on the name template.	• Parameters: name-java.lang.String
		• Returns: java.lang.String
			Example usage:
			Using Jacl:
			`$AdminControl queryNames WebSphere:type=Server,*`
			Using Jython:
			`AdminControl.queryNames('WebSphere:type=Server,*')`
			Example output:
			`WebSphere:cell=BaseApplicationServerCell,`
			` name=server1,mbeanIdentifier=server1,type=Server,`
			` node=mynode,process=server1`

queryNames_jmx	Returns a set of ObjectName objects that are based on the ObjectName object and the QueryExp query that you provide.	• Parameters: name-javax.management.ObjectName;query-javax.management.QueryExp
		• Returns: java.util.Set
			Example usage:
			Using Jacl:
			`set objectNameString [$AdminControl`
			` completeObjectName type=Server,*]`
			`set objName [$AdminControl`
			` makeObjectName $objNameString]`
			`set null [java::null]`
			`$AdminControl queryNames_jmx $objName $null`
			Using Jython:
			`objectNameString =`
			` AdminControl.completeObjectName`
			` ('type=Server,*')`
			`objName = AdminControl.makeObjectName`
			` (objectNameString)`
			`AdminControl.queryNames_jmx(objName,`
			` None)`
			Example output:
			`[WebSphere:cell=BaseApplicationServerCell,`
			`name=server1,mbeanIdentifier=server1,type=`
			`Server,node=mynode,process=server1]`

reconnect	Reconnects to the server, and clears information out of the local cache.	• Parameters: None
		• Returns: None
			Example usage:
			Using Jacl:
			`$AdminControl reconnect`
			Using Jython:
			`AdminControl.reconnect()`
			Example output:
			`WASX7074I: Reconnect of SOAP connector to`
			`host myhost completed.`

setAttribute	Sets the attribute value for the name that you provide.	• Parameters: name-java.lang.String; attributeName-java.lang.String; attributeValue-java.lang.String • Returns: None	Example usage: Using Jacl: set objNameString [$AdminControl completeObjectName WebSphere:type= TraceService,*] $AdminControl setAttribute $objNameString traceSpecification com.ibm.*=all=disabled Using Jython: objNameString = AdminControl.completeObjectName ('WebSphere:type=TraceService,*') AdminControl.setAttribute(objNameString, 'traceSpecification', 'com.ibm.*=all=disabled')
setAttribute_jmx	Sets the attribute value for the name that you provide.	• Parameters: name-ObjectName; attribute-javax.management.Attribute • Returns: None	Example usage: Using Jacl: set objectNameString [$AdminControl completeObjectName WebSphere:type= TraceService,*] set objName [$AdminControl makeObjectName $objectNameString] set attr [java::new javax.management.Attribute traceSpecification com.ibm.*=all=disabled] $AdminControl setAttribute_jmx $objName $attr Using Jython: objectNameString = AdminControl.completeObjectName ('WebSphere:type=TraceService,*') import javax.management as mgmt objName = AdminControl.makeObjectName (objectNameString) attr = mgmt.Attribute('traceSpecification', 'com.ibm.*=all=disabled') AdminControl.setAttribute_jmx(objName, attr)
setAttributes	Sets the attribute values for the names that you provide and returns a list of successfully set names.	• Parameters using Jacl: name-String; attributes-java.lang.String • Parameters using Jython: name-String; attributes-java.lang.String or name-String; attributes-java.lang.Object[] • Returns: java.lang.String	Example usage: Using Jacl: set objNameString [$AdminControl completeObjectName WebSphere:type= TracesService,*] $AdminControl setAttributes $objNameString {{traceSpecification com.ibm.ws.*=all=enabled}} Using Jython with string attributes: objNameString = AdminControl.completeObjectName ('WebSphere:type=TracesService,*') AdminControl.setAttributes(objNameString, '[[traceSpecification "com.ibm.ws.*=all=enabled"]]') Using Jython with object attributes: objNameString = AdminControl.completeObjectName ('WebSphere:type=TracesService,*') 473 AdminControl.setAttributes(objNameString, [['traceSpecification', 'com.ibm.ws.*=all=enabled']])

setAttributes_jmx	Sets the attribute values for the names that you provide and returns a list of successfully set names.	• Parameters: name-ObjectName; attributes-javax.management.AttributeList
		• Returns: javax.management.AttributeList
			Example usage:
			Using Jacl:
			`set objectNameString [$AdminControl`
			` completeObjectName WebSphere:type=TraceService,*]`
			`set objName [$AdminControl makeObjectName`
			` $objectNameString]`
			`set attr [java::new javax.management.Attribute`
			` traceSpecification com.ibm.ws.*=all=enabled]`
			`set alist`
			` [java::new javax.management.AttributeList]`
			`$alist add $attr`
			`$AdminControl setAttributes_jmx $objName $alist`
			Using Jython:
			`objectNameString =`
			` AdminControl.completeObjectName`
			` ('WebSphere:type=TraceService,*')`
			`import  javax.management  as  mgmt`
			`objName = AdminControl.makeObjectName`
			` (objectNameString)`
			`attr = mgmt.Attribute('traceSpecification',`
			` 'com.ibm.ws.*=all=enabled')`
			`alist = mgmt.AttributeList()`
			`alist.add(attr)`
			`AdminControl.setAttributes_jmx(objName,`
			` alist)`

startServer	Starts the specified application server by locating it in the configuration. This command uses the default wait time. You can only use this command if the scripting client is connected to a node agent. This command returns a message to indicate if the server starts successfully.	• Parameters: server name-java.lang.String
		• Returns: java.lang.String
			Example usage:
			Using Jacl:
			`$AdminControl startServer server1`
			Using Jython:
			`AdminControl.startServer('server1')`

startServer	Starts the specified application server by locating it in the configuration. The start process waits the number of seconds specified by the wait time for the server to start. You can only use this command if the scripting client is connected to a node agent. This command returns a message to indicate if the server starts successfully.	• Parameters: server name-java.lang.String, wait time-java.lang.String
		• Returns: java.lang.String
			Example usage:
			Using Jacl:
			`$AdminControl startServer server1 100`
			Using Jython:
			`AdminControl.startServer('server1', 100)`

startServer	Starts the specified application server by locating it in the configuration. This command uses the default wait time. You can use this command when the scripting client is either connected to a node agent or to a deployment manager process. It returns a message to indicate if the server starts successfully.	• Parameters: server name-java.lang.String, node name-java.lang.String
		• Returns: java.lang.String
			Example usage:
			Using Jacl:
			`$AdminControl startServer server1 myNode`
			Using Jython:
			`AdminControl.startServer('server1', 'myNode')`

startServer	Starts the specified application server by locating it in the configuration. The start process waits the number of seconds specified by the wait time for the server to start. You can use this command when the scripting client is either connected to a node agent or to a deployment manager process. This command returns a message to indicate if the server starts successfully.	• Parameters: server name-java.lang.String, node name-java.lang.String, wait time-java.lang.String
		• Returns: java.lang.String
			Example usage:
			Using Jacl:
			`$AdminControl startServer server1 myNode 100`
			Using Jython:
			`AdminControl.startServer('server1', 'myNode', 100)`

stopServer	Stops the specified application server. The command returns a message to indicate if the server stops successfully.	• Parameters: server name-java.lang.String
		• Returns: java.lang.String
			Example usage:
			Using Jacl:
			`$AdminControl stopServer server1`
			Using Jython:
			`AdminControl.stopServer('server1')`

stopServer	Stops the specified application server. If you set the flag to `immediate`, the server stops immediately. Otherwise, a normal stop occurs. This command returns a message to indicate if the server stops successfully.	• Parameters: server name-java.lang.String, immediate flag-java.lang.String
		• Returns: java.lang.String
			Example usage:
			Using Jacl:
			`$AdminControl stopServer server1 immediate`
			Using Jython:
			`AdminControl.stopServer('server1', 'immediate')`

stopServer	Stops the specified application server. This command returns a message to indicate if the server stops successfully.	• Parameters: server name-java.lang.String, node name-java.lang.String
		• Returns: java.lang.String
			Example usage:
			Using Jacl:
			`$AdminControl stopServer server1 myNode`
			Using Jython:
			`AdminControl.stopServer('server1', 'my Node')`

stopServer	Stops the specified application server. If you set the flag to `immediate`, the server stops immediately. Otherwise, a normal stop occurs. This command returns a message to indicate if the server stops successfully.	• Parameters: server name-java.lang.String, node name-java.lang.String, immediate flag-java.lang.String
		• Returns: java.lang.String
			Example usage:
			Using Jacl:
			`$AdminControl stopServer server1 myNode immediate`
			Using Jython:
			`AdminControl.stopServer('server1', 'my Node', 'immediate')`

testConnection	A convenience command communicates with the DataSourceCfgHelper MBean to test a DataSource connection. This command works with the DataSource that resides in the configuration repository. If the DataSource to be tested is in the temporary workspace that holds the update to the repository, you have to save the update to the configuration repository before running this command. Use this command with the configuration ID that corresponds to the DataSource and the WAS40DataSource object types. The return value is a message that contains the message indicating a successful connection or a connection with warning. If the connection fails, an exception is created from the server indicating the error.	• Parameters: configId-java.lang.String
		• Returns: java.lang.String
			Example usage:
			Using Jacl:
			`set ds [lindex [$AdminConfig list DataSource] 0]`
			`$AdminControl testConnection $ds`
			Using Jython:
			`# get line separator`
			`import java.lang.System as sys`
			`lineSeparator = sys.getProperty('line.separator')`
			`ds = AdminConfig.list('DataSource').split`
			` (lineSeparator)[0]`
			`AdminControl.testConnection(ds)`
			Example output:
			`WASX7217I: Connection to provided`
			`datasource was successful.`

testConnection	Deprecated.
	This command can give false results and does not work when connected to a node agent. As of V5.0.2, the preferred way to test a data source connection is with the **testConnection** command that passes in the DataSource configId parameter as the only parameter.
		- Parameters: configId-java.lang.String; props-java.lang.String
		- Returns: java.lang.String
			Example usage:
			Using Jacl:
			`set ds [lindex [$AdminConfig list DataSource] 0]`
			`$AdminControl testConnection $ds {{prop1 val1}}`
			Using Jython:
			`# get line separator`
			`import java.lang.System as sys`
			`lineSeparator = sys.getProperty('line.separator')`
			`ds = AdminConfig.list('DataSource').split`
			` (lineSeparator)[0]`
			`AdminControl.testConnection(ds, '[[prop1 val1]]')`
			Example output:
			`WASX7390E: Operation not supported -`
			`testConnection command with config id`
			`and properties arguments is not supported.`
			`Use testConnection command with`
			`config id argument only.`

trace	Sets the trace specification for the scripting process to the value that you specify.	- Parameters: traceSpec-java.lang.String
		- Returns: None
			Example usage:
			Using Jacl:
			`$AdminControl trace com.ibm.ws.scripting.*=`
			` all=enabled`
			Using Jython:
			`AdminControl.trace('com.ibm.ws.scripting.*=`
			` all=enabled')`

Commands for the AdminApp object

Use the AdminApp object to install, modify, and administer applications. The AdminApp object interacts with the WebSphere Application Server management and configuration services to make application inquiries and changes. This interaction includes installing and uninstalling applications, listing modules, exporting, and so on.

You can start the scripting client when no server is running, if you want to use only local operations. To run in local mode, use the `-conntype NONE` option to start the scripting client. You receive a message that you are running in the local mode. Running the AdminApp object in local mode when a server is currently running is not recommended. This is because any configuration changes made in local mode will not be reflected in the running server configuration and vice versa. If you save a conflicting configuration, you could corrupt the configuration. In a deployment manager environment, configuration updates are available only if a scripting client is connected to a deployment manager. When connected to a node agent or a managed application server, you will not be able to update the configuration because the configuration for these server processes are copies of the master configuration which resides in the deployment manager. The copies are created on a node machine when a configuration synchronization occurs between the deployment manager and the node agent. Make configuration changes to the server processes by connecting a scripting client to a deployment manager. For this reason, to change a configuration, do not run a scripting client in local mode on a node machine. It is not a supported configuration.

The following commands are available for the AdminApp object:

Command name:	Description:	Parameters and return values:	Examples:

deleteUserAndGroupEntries	Deletes users or groups for all roles, and deletes user IDs and passwords for all of the RunAs roles that are defined in the application.	• Parameters: appname
		• Returns: None
			Example usage:
			Using Jacl:
			`$AdminApp deleteUserAndGroupEntries myapp`
			Using Jython:
			`AdminApp.deleteUserAndGroupEntries('myapp')`

edit	Edits an application or module in non-interactive mode.
	The **edit** command changes the application deployment. Specify these changes in the options parameter. No options are required for the **edit** command.
		• Parameters using Jacl: appname - string; options - string
		• Parameters using Jython: appname - string; options - string or appname - string; options - Jython list
		• Returns: string
			Example usage:
			Using Jacl:
			`$AdminApp edit "JavaMail Sample"`
			` {-MapWebModToVH {{"JavaMail`
			`Sample WebApp" mtcomps.war,WEB-INF/web.xml`
			` newVH}}}`
			Using Jython with string options:
			`AdminApp.edit("JavaMail Sample",`
			` '[-MapWebModToVH [["JavaMail`
			`32 Sample WebApp" mtcomps.war,WEB-INF/web.xml`
			` newVH]]]')`
			Using Jython with list options:
			`option = [["JavaMail 32 Sample WebApp",`
			` "mtcomps.war,WEB-INF/web.xml",`
			`"newVH"]]`
			`mapVHOption = ["-MapWebModToVH", option]`
			`AdminApp.edit("JavaMail Sample", mapVHOption)`

editInteractive	Edits an application or module in interactive mode.
	The **editInteractive** command changes the application deployment. Specify these changes in the options parameter. No options are required for the **editInteractive** command.
		• Parameters using Jacl: appname - string; options - string
		• Parameters using Jython: appname - string; options - string or appname - string; options - Jython list
		• Returns: string
			Example usage:
			Using Jacl:
			`$AdminApp editInteractive ivtApp`
			Using Jython:
			`AdminApp.editInteractive('ivtApp')`

export	Exports the application appname parameter to a file that you specify by file name.	• Parameters: appname, filename
		• Returns: None
			Example usage:
			Using Jacl:
			`$AdminApp export "My App" /usr/me/myapp.ear`
			Using Jython:
			`AdminApp.export("My App", '/usr/me/myapp.ear')`

exportDDL	Extracts the data definition language (DDL) from the application appname parameter to the directoryname parameter that a directory specifies. The options parameter is optional.	• Parameters: appname, directoryname, options
		• Returns: None
			Example usage:
			Using Jacl:
			`$AdminApp exportDDL "My App" /usr/me/DDL {-ddlprefix myApp}`
			Using Jython:
			`AdminApp.exportDDL("My App", '/usr/me/DDL', '[-ddlprefix myApp]')`

help	Displays general help for the AdminApp object.	- Parameters: None
		- Returns: None
			Example usage:
			Using Jacl:
			`$AdminApp help`
			Using Jython:
			`print AdminApp.help()`
			Example output:
			`WASX7095I: The AdminApp object allows`
			`application objects to be manipulated`
			`including installing, uninstalling,`
			`editing,and listing.  Most of the commands`
			`supported by AdminApp operate in two modes:`
			`the default mode is one in which AdminApp`
			`communicates with the WebSphere Application`
			`Server to accomplish its tasks.  A local`
			`mode is also possible, in which no server`
			`communication takes place.  The local mode`
			`of operation is invoked by including the`
			`"-conntype NONE" flag in the option string`
			`supplied to the command.`
			
			`The following commands are supported by`
			`AdminApp; more detailed information about`
			`each of these commands is available by`
			`using the "help" command of AdminApp`
			`and supplying the name of the command`
			`as an argument.`
			
			`edit            Edit the properties of an`
			`       application`
			`editInteractive Edit the properties of an`
			`       application interactively`
			`export          Export application to a file`
			`exportDDL      Extract DDL from application`
			`        to a directory`
			`help            Show help information`
			`install         Installs an application,`
			`       given a file name and an`
			`       option string.`
			`installInteractive`
			`                Installs an application`
			`       in interactive mode, given`
			`       a file name and an option`
			`       string.`
			`list            List all installed`
			`       applications`
			`listModules     List the modules in a`
			`       specified application`
			`options         Shows the options available,`
			`       either for a given file,`
			`       or in general.`
			`taskInfo        Shows detailed information`
			`       pertaining to a given`
			`       installation task for a`
			`       given file`
			`uninstall       Uninstalls an application,`
			`       given an application name`
			`                and an option string`

help	Displays help for an AdminApp command or installation option.	• Parameters: operation name
		• Returns: none
			Example usage:
			Using Jacl:
			`$AdminApp help uninstall`
			Using Jython:
			`print AdminApp.help('uninstall')`
			Example output:
			`WASX7102I: Method: uninstall`
			`Arguments: application name, options`
			`Description: Uninstalls application named`
			`by "application name" using the options`
			`supplied by String 2.`
			`Method: uninstall`
			`Arguments: application name`
			`Description: Uninstalls the application`
			`specified by`
			`"application name" using default options.`

install	Installs an application in non-interactive mode, given a fully qualified file name and a string of installation options. The options parameter is optional.	• Parameters using Jacl: earfile- string; options- string
		• Parameters using Jython: earfile- string; options- string or earfile- string; options- Jython list
		• Returns: None
			Example usage:
			Using Jacl:
			`$AdminApp install c:/apps/myapp.ear`
			Using Jython:
			`AdminApp.install('c:/apps/myapp.ear')`
			Many options are available for this command. You can obtain a list of valid options for an Enterprise Archive (EAR) file with the following command:
			Using Jacl:
			`$AdminApp options myApp.ear`
			Using Jython:
			`AdminApp.options('myApp.ear')`
			You can also obtain help for each object with the following command:
			Using Jacl:
			`$AdminApp help MapModulesToServers`
			Using Jython:
			`AdminApp.help('MapModulesToServers')`

installInteractive	Installs an application in interactive mode, given a fully qualified file name and a string of installation options. The options parameter is optional.	• Parameters using Jacl: earfile- string; options- string
		• Parameters using Jython: earfile- string; options- string or earfile- string; options- Jython list
		• Returns: None
			Example usage:
			Using Jacl:
			`$AdminApp installInteractive c:/websphere/`
			`appserver/installableApps/jmsample.ear`
			Using Jython:
			`AdminApp.installInteractive('c:/websphere/`
			`appserver/installableApps/jmsample.ear')`

list	Lists the applications that are installed in the configuration.	- Parameters: None
		- Returns: application names
			Example usage:
			Using Jacl:
			`$AdminApp list`
			Using Jython:
			`print AdminApp.list()`
			Example output:
			`adminconsole`
			`DefaultApplication`
			`ivtApp`

listModules	Lists the modules in an application.
	The options parameter is optional. The valid option is -server. This option lists the application servers on which the modules are installed.
		- Parameters: appname, options
		- Returns: modules in the application
			Example usage:
			Using Jacl:
			`$AdminApp listModules ivtApp`
			Using Jython:
			`print AdminApp.listModules('ivtApp')`
			Example output:
			`ivtApp#ivtEJB.jar+META-INF/ejb-jar.xml`
			`ivtApp#ivt_app.war+WEB-INF/web.xml`
			This example is formed by the concatenation of appname, #, module URI, +, and DD URI. You can pass this string to the **edit** and **editInteractive** AdminApp commands.

options	Displays a list of options for installing an Enterprise Archive (EAR) file.	• Parameters: earfile • Returns: Information about the valid installation options for an Enterprise Archive (EAR) file.	Example usage: Using Jacl: $AdminApp options c:/websphere/appserver/installableApps/ ivtApp.ear Using Jython: AdminApp.options ('c:/websphere/appserver/installableApps/ ivtApp.ear') Example usage: WASX7112I: The following options are valid for "c:/websphere/appserver/installableapps/ ivtApp.ear" MapRolesToUsers BindJndiForEJBNonMessageBinding MapEJBRefToEJB MapWebModToVH MapModulesToServers EnsureMethodProtectionFor10EJB GetServerName preCompileJSPs nopreCompileJSPs distributeApp nodistributeApp useMetaDataFromBinary nouseMetaDataFromBinary deployejb nodeployejb createMBeansForResources nocreateMBeansForResources reloadEnabled noreloadEnabled deployws nodeployws usedefaultbindings defaultbinding.force allowPermInFilterPolicy noallowPermInFilterPolicy verbose update update.ignore.old update.ignore.new installed.ear.destination appname reloadInterval validateinstall deployejb.rmic deployejb.dbtype deployejb.dbschema deployejb.classpath deployws.classpath deployws.jardirs defaultbinding.datasource.jndi defaultbinding.datasource.username defaultbinding.datasource.password defaultbinding.cf.jndi defaultbinding.cf.resauth defaultbinding.ejbjndi.prefix defaultbinding.virtual.host defaultbinding.strategy.file server node cell cluster contextroot custom

options	Displays a list of options for editing an existing application.	• Parameters: Application name
		• Returns: Information about the valid edit options for an application.
			Example usage:
			Using Jacl:
			`$AdminApp options ivtApp`
			Using Jython:
			`AdminApp.options('ivtApp')`
			Example output:
			`WASX7112I: The following options are valid for "ivtApp"`
			`MapRolesToUsers`
			`BindJndiForEJBNonMessageBinding`
			`MapEJBRefToEJB`
			`MapWebModToVH`
			`MapModulesToServers`
			`distributeApp`
			`nodistributeApp`
			`useMetaDataFromBinary`
			`nouseMetaDataFromBinary`
			`createMBeansForResources`
			`nocreateMBeansForResources`
			`reloadEnabled`
			`noreloadEnabled`
			`verbose`
			`installed.ear.destination`
			`reloadInterval`

options	Displays a list of options for editing a module in an existing application.	• Parameters: application module name. This parameter requires the same module name format as the output that is returned by the **listModules** command.
		• Returns: Information about the valid edit options for a module.
			Example usage:
			Using Jacl:
			`$AdminApp options ivtApp#ivtEJB.jar+META-INF/ejb-jar.xml`
			Using Jython:
			`AdminApp.options('ivtApp#ivtEJB.jar+META-INF/ejb-jar.xml')`
			Example output:
			`WASX7112I: The following options are valid for "ivtApp#ivtEJB.jar+META-INF/ejb-jar.xml"`
			`MapRolesToUsers`
			`BindJndiForEJBNonMessageBinding`
			`MapModulesToServers`
			`verbose`

options	Displays a list of options for installing or updating an application or application module file.	• Parameters:
		file, operation - The following list includes the valid values:
		– installapp - Installing the file that is specified
		– updateapp - Updating an existing application with the file that is specified
		– addmodule - Adding the module file that is specified to an existing application
		– updatemodule - Updating an existing module in an application with the module file that is specified
		• Returns: Information about the valid options that are available for the operation that is requested with the input file.
			Example using the updateapp operation:
			Using Jacl:
			`$AdminApp options`
			` c:/websphere/appserver/installableApps/`
			`ivtApp.ear updateapp`
			Using Jython:
			`AdminApp.options`
			` ('c:/websphere/appserver/installableApps/`
			`ivtApp.ear', 'updateapp')`
			Example using the addmodule operation:
			Using Jacl:
			`$AdminApp options myModule.jar addmodule`
			Using Jython:
			`AdminApp.options`
			` ('DefaultWebApplication.war', 'addmodule')`
			Example output using the updateapp operation:
			`WASX7112I: The following options are valid`
			`for "c:/websphere/appserver/installableApps/`
			`ivtApp.ear"`
			`MapRolesToUsers`
			`BindJndiForEJBNonMessageBinding`
			`MapEJBRefToEJB`
			`MapWebModToVH`
			`MapModulesToServers`
			`EnsureMethodProtectionFor10EJB`
			`GetServerName`
			`preCompileJSPs`
			`nopreCompileJSPs`
			`distributeApp`
			`nodistributeApp`
			`useMetaDataFromBinary`
			`nouseMetaDataFromBinary`
			`deployejb`
			`nodeployejb`
			`createMBeansForResources`
			`nocreateMBeansForResources`
			`reloadEnabled`
			`noreloadEnabled`
			`deployws`
			`nodeployws`
			`usedefaultbindings`
			`defaultbinding.force`
			`allowPermInFilterPolicy`
			`noallowPermInFilterPolicy`
			`verbose`
			`update`
			`update.ignore.old`
			`update.ignore.new`
			`installed.ear.destination`
			`reloadInterval`
			`deployejb.rmic`
			`deployejb.dbtype`
			`deployejb.dbschema`
			`deployejb.classpath`
			`deployws.classpath`
			`deployws.jardirs`

options continued			defaultbinding.datasource.jndi defaultbinding.datasource.username defaultbinding.datasource.password defaultbinding.cf.jndi defaultbinding.cf.resauth defaultbinding.ejbjndi.prefix defaultbinding.virtual.host defaultbinding.strategy.file appname contextroot custom contenturi contents operation Example output using the addmodule operation: WASX7112I: The following options are valid for "DefaultWebApplication.war" MapRolesToUsers MapEJBRefToEJB MapWebModToVH MapModulesToServers GetServerName preCompileJSPs nopreCompileJSPs deployejb nodeployejb deployws nodeployws usedefaultbindings defaultbinding.force verbose defaultbinding.datasource.jndi defaultbinding.datasource.username defaultbinding.datasource.password defaultbinding.cf.jndi defaultbinding.cf.resauth defaultbinding.ejbjndi.prefix defaultbinding.virtual.host defaultbinding.strategy.file server node cell cluster contextroot custom contenturi contents operation
publishWSDL	Publishes Web Services Description Language (WSDL) files for the application that is specified in the appname parameter to the file that is specified in the filename parameter.	• Parameters: appname, filename • Returns: None	Example usage: Using Jacl: $AdminApp publishWSDL JAXRPCHandlerServer c:/temp/a.zip Using Jython: AdminApp.publishWSDL ('JAXRPCHandlerServer', 'c:/temp/a.zip')

publishWSDL	Publishes Web Services Description Language (WSDL) files for the application that is specified in the appname parameter to the file that is specified in the filename parameter using the SOAP address prefixes that are specified in the soapAddressPrefixes parameter.	• Parameters: appname, filename, soapAddressPrefixes
		• Returns: None
			Example usage:
			Using Jacl:
			`$AdminApp publishWSDL JAXRPCHandlersServer`
			` c:/temp/a.zip`
			`{{JAXRPCHandlersServerApp.war`
			`  {{http http://localhost:9080}}}}`
			Using Jython:
			`AdminApp.publishWSDL('JAXRPCHandlersServer',`
			` 'c:/temp/a.zip',`
			`'[[JAXRPCHandlersServerApp.war`
			` [[http http://localhost:9080]]]]')`

searchJNDI References	Lists applications that refer to the Java Naming and Directory Interface (JNDI) name on a specific node.	• Parameters: Node configuration ID, options
		• Returns: string
			Example usage:
			The following example assumes that an installed application named `MyApp` has a JNDI name of `eis/J2CCF1`.
			Using Jacl:
			`$AdminApp searchJNDIReferences $node`
			` {-JNDIName eis/J2CCF1 -verbose}`
			Using Jython:
			`print AdminApp.searchJNDIReferences(node,`
			` '[-JNDIName eis/J2CCF1 -verbose]')`
			Example output:
			`WASX7410W: This operation may take a while`
			`depending on the number of applications`
			`installed in your system.`
			`MyApp`
			`MapResRefToEJB :ejb-jar-ic.jar : [eis/J2CCF1]`

taskInfo	Provides information about a particular task option for an application file.	• Parameters: earfile, task name
		• Returns: None
			Example usage:
			Using Jacl:
			`$AdminApp taskInfo`
			` c:/websphere/appserver/installableApps/`
			`jmsample.ear MapWebModToVH`
			Using Jython:
			`print AdminApp.taskInfo`
			` ('c:/websphere/appserver/installableApps/`
			`jmsample.ear', 'MapWebModToVH')`
			Example output:
			`MapWebModToVH: Selecting virtual hosts for`
			`Web modules Specify the virtual host where`
			`you want to install the Web modules that are`
			`contained in your application. Web modules`
			` can be installed on the same virtual host`
			`or dispersed among several hosts. Each`
			`element of the MapWebModToVH task consists`
			`of the following three fields: "webModule,"`
			` "uri," "virtualHost." Of these fields,`
			`the following fields might be assigned`
			`new values: "virtualHost"and the following`
			`are required: "virtualHost"`
			
			`The current contents of the task after`
			`running default bindings are:`
			`webModule: JavaMail Sample WebApp`
			`uri: mtcomps.war,WEB-INF/web.xml`
			`virtualHost: default_host`

uninstall	Uninstalls an existing application.	• Parameters: appname- string
		• Returns: None
			Example usage:
			Using Jacl:
			`$AdminApp uninstall myApp`
			Using Jython:
			`AdminApp.uninstall('myApp')`
			Example output:
			`ADMA5017I: Uninstallation of myapp started.`
			`ADMA5104I: Server index entry for`
			`myCellManager was updated successfully.`
			`ADMA5102I: Deletion of config data for myapp`
			`from config repository completed successfully.`
			`ADMA5011I: Cleanup of temp dir for app myapp`
			`done.`
			`ADMA5106I: Application myapp uninstalled`
			`successfully.`

updateAccessIDs	Updates the access ID information for users and groups that are assigned to various roles that are defined in the application. The access IDs are read from the user registry and saved in the application bindings. This operation improves run-time performance of the application. Call this command after installing an application or after editing security role-specific information for an installed application. This method cannot be invoked when the -conntype option is set to `NONE`. You must be connected to a server to invoke this command.
	The bALL Boolean parameter retrieves and saves all access IDs for users and groups in the application bindings. Specify `false` if you want to retrieve access IDs for users or groups that do not have an access ID in the application bindings.
		• Parameters: appname, bALL
		• Returns: None
			Example usage:
			Using Jacl:
			`$AdminApp updateAccessIDs myapp true`
			Using Jython:
			`AdminApp.updateAccessIDs('myapp', 'true')`

view	View the task that is specified by the taskname option parameter for the application or by the module that is specified by the name parameter. Use -tasknames as the option to get a list of valid task names for the application. Otherwise, specify one or more task names as the option.	• Parameters: name, taskname option
		• Returns: string
			Example usage:
			Using Jacl:
			`$AdminApp view adminconsole {-tasknames}`
			Using Jython:
			`AdminApp.view('adminconsole', ['-tasknames'])`
			Example output:
			`MapModulesToServers`
			`MapWebModToVH`
			`MapRolesToUsers`
			Using Jacl:
			`$AdminApp view adminconsole`
			` {-MapModulesToServers}`
			Using Jython:
			`AdminApp.view('adminconsole',`
			` ['-MapModulesToServers'])`
			Example output:
			`MapModulesToServers: Selecting Application`
			` Servers`
			
			`Specify the application server where you`
			`want to install the modules that are`
			`contained in your application. Modules can`
			`be installed on the same server or dispersed`
			`among several servers:`
			
			`Module:  adminconsole`
			`URI:  adminconsole.war,WEB-INF/web.xml`
			`Server:  WebSphere:cell=juniartiNetwork,`
			`node=juniartiManager,server=dmgr`
			Example usage:
			Using Jacl:
			`$AdminApp view adminconsole#adminconsole.war+`
			`WEB-INF/web.xml {-MapRolesToUsers}`
			Using Jython:
			`AdminApp.view`
			` ('adminconsole#adminconsole.war+WEB-INF/`
			`web.xml',  ['-MapRolesToUsers'])`
			Example output:
			`MapRolesToUsers: Mapping Users to Roles`
			
			`Each role that is defined in the application`
			`or the module must be mapped to a user or a`
			`group from the user registry of the domain:`
			
			`Role:  administrator`
			`Everyone?:  No`
			`All Authenticated?:  No`
			`Mapped Users:`
			`Mapped Groups:`

view continued			Role: operator Everyone?: No All Authenticated?: No Mapped Users: Mapped Groups: Role: configurator Everyone?: No All Authenticated?: No Mapped Users: Mapped Groups: Role: monitor Everyone?: No All Authenticated?: No Mapped Users: Mapped Groups:

update	Updates an application in non-interactive mode. Provide the application name, content type, and update options.	• Parameters using Jacl: appname, content type, options – string format
		• Parameters using Jython: appname, content type, option- string or list format
		• Returns: String
		This command supports the addition, removal, and update of application subcomponents or the entire application.
		Use the content type parameter to indicate if you want to update part of the application or the entire application. The following list includes the valid content type values for the **update** command:
		– `app` - Indicates that you want to update the entire application. This option is the same as indicating the update option with the **install** command. With the `app` value as the content type, you must specify the operation option with update as the value. Provide the new enterprise archive file (EAR) file using the contents option. You can also specify binding information and application options. By default, binding information for installed modules is merged with the binding information for updated modules. To change this default behavior, specify the update.ignore.old or the update.ignore.new options.
			Example usage:
			Using Jacl:
			`$AdminApp update myApp file {-operation`
			`add -contents`
			`c:/apps/myApp/web.xml`
			` -contenturi META-INF/web.xml}`
			Using Jython with string options:
			`AdminApp.update('myApp', 'file',`
			`'[-operation add -contents`
			`c:/apps/myApp/web.xml`
			`-contenturi META-INF/web.xml]')`
			Using Jython with list options:
			`AdminApp.update('myApp', 'file',`
			`['-operation', 'add',`
			`'-contents',  'c:/apps/myApp/web.xml',`
			`'-contenturi',`
			`'META-INF/web.xml'])`
			Example output:
			`Update of singleFile has started.`
			`ADMA5009I: Application archive extracted at`
			`C:\DOCUME~1\lavena\LOCALS~1\Temp\`
			` app_fb5a1960f0\ext`
			`Added files from partial ear: []`
			`performFileOperation:`
			`source=C:\DOCUME~1\lavena\LOCALS~1\Temp\`
			`app_fb5a1960f0\ext, dest=C:\DOCUME~1\lavena\`
			`LOCALS~1\Temp\`
			`app_fb5a1960f0\mrg, uri= META-INF/web.xml,`
			`op= add`
			`Copying file from C:\DOCUME~1\lavena\`
			`LOCALS~1\Temp\app_fb5a1960f0\ext/`
			`META-INF/web.xml to`
			`C:\DOCUME~1\lavena\LOCALS~1\Temp`
			`\app_fb5a1960f0\mrg\META-INF\web.xml`
			`Collapse list is: []`
			`FileMergeTask completed successfully`
			`ADMA5005I: Application singleFile`
			`configured in WebSphere repository`
			`delFiles: []`
			`delM: null`
			`addM: null`

update continued		• `file` - Indicates that you want to update a single file. You can add, remove, or update individual files at any scope within the deployed application. With the `file` value as the content type, you must perform operations on the file using the `operation` option. Depending on the type of operation, additional options are required. For file additions and updates, you must provide file content and the file URI relative to the root of the EAR file using the contents and contenturi options. For file deletion, you must provide the file URI relative to the root of the EAR file using the contenturi option which is the only required input. Any other options that you provide are ignored.	`Pattern for remove loose and mod:`
			`Loose add pattern: META-INF/[^/]*|`
			`WEB-INF/[^/]*|.*wsdl`
			`root file to be copied: META-INF/web.xml to`
			`C:\asv\b0403.04\WebSphere\AppServer\`
			`wstemp\Scriptfb5a191b4e\workspace\cells\`
			`BAMBIE\applications\`
			`singleFile.ear\deployments\singleFile/`
			`META-INF/web.xml`
			`ADMA5005I: Application singleFile`
			`configured in WebSphere repository`
			`xmlDoc: [#document: null]`
			`root element: [app-delta: null]`
			`****** delta file name: C:\asv\b0403.04\`
			`WebSphere\AppServer\wstemp\Scriptfb5a191`
			`b4e\workspace\cells\BAMBIE\applications\`
			`singleFile.ear/deltas/delta-1079548405564`
			`ADMA5005I: Application singleFile`
			`configured in WebSphere repository`
			`ADMA6011I: Deleting directory tree`
			`C:\DOCUME~1\lavena\LOCALS~1\Temp\app_fb5a1960f0`
			`ADMA5011I: Cleanup of temp dir for app`
			`singleFile done.`
			`Update of singleFile has ended.`

update continued		• modulefile - Indicates that you want to update a module. You can add, remove, or update an individual application module. If you specify the modulefile value as the content type, you must indicate the operation that you want to perform on the module using the operation option. Depending on the type of operation, further options are required. For installing new modules or updating existing modules in an application, you must indicate the file content and the file URI relative to the root of the EAR file using the contents and contenturi options. You can also specify binding information and application options that pertain to the new or updated modules. For module updates, the binding information for the installed module is merged with the binding information for the input module by default. To change the default behavior, specify the update.ignore.old or the update.ignore.new options. To delete a module, indicate the file URI relative to the root of the EAR file.	

update continued		- `partialapp` - Indicates that you want to update a partial application. Using a subset of application components provided in a zip file format you can update, add, and delete files and modules. The zip file is not a valid Java 2 platform, Enterprise Edition (J2EE) archive. Instead, it contains application artifacts in the same hierarchical structure as they display in an EAR file. For more information on how to construct the partial application zip file, see the Java API section. If you indicate the `partialapp` value as the content type, use the contents option to specify the location of the zip file. When a partial application is provided as an update input, binding information and application options cannot be specified and are ignored, if provided. For a list of the valid options for the **update** command, see "Options for the AdminApp object install, installInteractive, edit, editInteractive, update, and updateInteractive commands" on page 437.	

updateInteractive	Updates an application in interactive mode. Provide the application name, content type, and update options.	- Parameters using Jacl: appname, content type, options - string format
		- Parameters using Jython: appname, content type, option - string or list format
		- Returns: String
		Use the **updateInteractive** command to add, remove, and update application subcomponents or an entire application. When you update an application module or an entire application using interactive mode, the steps that you use to configure binding information are similar to those that apply to the **installInteractive** command. If you update a file or a partial application, the steps that you use to configure the binding information are not available. In this case, the steps are the same as the ones you use with the **update** command.
		Use the content type parameter to indicate if you want to update part of the application or the entire application. The following list contains the valid content type values for the **updateInteractive** command:
		– `app` - Indicates that you want to update the entire application. This option is the same as indicating the update option with **install** command. With the `app` value as the content type, you must specify the operation option with update as the value. Provide the new enterprise archive file (EAR) file using the contents option. You can also specify binding information and application options. By default, binding information for installed modules is merged with the binding information for updated modules. To change this default behavior, specify the update.ignore.old or the update.ignore.new options.
			Example usage:
			Using Jacl:
			`$AdminApp updateInteractive myApp modulefile`
			`{-operation`
			`add -contents c:/apps/myApp/Increment.jar`
			`-contenturi`
			`Increment.jar -nodeployejb`
			` -BindJndiForEJBNonMessageBinding`
			`{{"Increment Enterprise JavaBeans"`
			` Increment Increment.`
			`jar,META-INF/ejb-jar.xml Inc}}}`
			Using Jython string:
			`AdminApp.updateInteractive('myApp',`
			`'modulefile', '[-operation add -contents`
			`c:/apps/myApp/Increment.jar -contenturi`
			`Increment.jar -nodeployejb`
			`-BindJndiForEJBNonMessageBinding`
			`[["Increment Enterprise JavaBeans"`
			`Increment Increment.jar,META-INF/`
			`ejb-jar.xml Inc]]]')`
			Using Jython list:
			`bindJndiForEJBValue = [["Increment Enterprise`
			`JavaBeans", "Increment", "Increment.jar,`
			`META-INF/ejb-jar.xml", "Inc"]]`
			
			`AdminApp.updateInteractive('myApp',`
			`'modulefile', ['-operation', 'add',`
			`'-contents',  'c:/apps/myApp/Increment.jar',`
			` '-contenturi', 'Increment.jar',`
			`'-nodeployejb',`
			`'-BindJndiForEJBNonMessageBinding',`
			`bindJndiForEJBValue])`
			Example output:
			`Getting tasks for: myApp`
			`WASX7266I: A was.policy file exists for this`
			`application; would you like to display it? [No]`
			
			`Task[4]: Binding enterprise beans to JNDI`
			`names Each non message driven enterprise`
			`bean in your application or module must be`
			`bound to a JNDI name.`
			
			`EJB Module:  Increment Enterprise Java Bean`
			`EJB:  Increment`
			`URI:  Increment.jar,META-INF/ejb-jar.xml`
			`JNDI Name:  [Inc]:`
			
			`Task[10]: Specifying the default data source`
			`for EJB 2.x modules`
			`Specify the default data source for`
			`the EJB 2.x Module containing 2.x CMP beans.`

updateInteractive continued		• file - Indicates that you want to update a single file. You can add, remove, or update individual files at any scope within the deployed application. With the file value as the content type, you must perform operations on the file using the operation option. Depending on the type of operation, additional options are required. For file additions and updates, you must provide file content and the file URI relative to the root of the EAR file using the contents and contenturi options. For file deletion, you must provide the file URI relative to the root of the EAR file using the contenturi option which is the only required input. Any other options that you provide are ignored.
		• modulefile - Indicates that you want to update a module. You can add, remove, or update an individual application module. If you specify the modulefile value as the content type, you must indicate the operation that you want to perform on the module using the operation option. Depending on the type of operation, additional options are required. For installing new modules or updating existing modules in an application, you must indicate the file content and the file URI relative to the root of the EAR file using the contents and the contenturi options. You can also specify binding information and application options that pertain to the new or updated modules. For module updates, the binding information for the installed module is merged with the binding information for the input module by default. To change the default behavior, specify the update.ignore.old or the update.ignore.new options. To delete a module, indicate the file URI relative to the root of the EAR file.
			`WASX7349I: Possible value for resource`
			`authorization is container or per connection`
			`factory`
			`EJB Module:  Increment Enterprise Java Bean`
			`URI:  Increment.jar,META-INF/ejb-jar.xml`
			`JNDI Name:  [DefaultDatasource]:`
			`Resource Authorization:  [Per connection`
			`factory]:`
			
			`Task[12]: Specifying data sources for`
			`individual 2.x CMP beans`
			`Specify an optional data source for each`
			`2.x CMP bean. Mapping a specific data source`
			`to a CMP bean`
			`overrides the default data source for`
			`the module containing the enterprise bean.`
			
			`WASX7349I: Possible value for resource`
			`authorization is container or per connection`
			`factory`
			`EJB Module:  Increment Enterprise Java Bean`
			`EJB:  Increment`
			`URI:  Increment.jar,META-INF/ejb-jar.xml`
			`JNDI Name:  [DefaultDatasource]:`
			`Resource Authorization:  [Per connection`
			`factory]: container`
			`Setting "Resource Authorization" to`
			`"cmpBinding.container"`
			`Task[14]: Selecting Application Servers`
			`Specify the application server where you want`
			`to install modules that are contained in your`
			`application.`
			`Modules can be installed on the same server or`
			`dispersed among several servers.`
			
			`Module:  Increment Enterprise Java Bean`
			`URI:  Increment.jar,META-INF/ejb-jar.xml`
			`Server:  [WebSphere:cell=myCell,node=myNode,`
			`server=server1]:`
			
			`Task[16]: Selecting method protections for`
			`unprotected methods for 2.x EJB`
			`Specify whether you want to assign security`
			`role to the unprotected method, add the`
			`method to the exclude list, or mark the`
			`method as unchecked.`
			
			`EJB Module:  Increment Enterprise Java Bean`
			`URI:  Increment.jar,META-INF/ejb-jar.xml`
			`Protection Type:  [methodProtection.uncheck]:`
			
			`Task[18]: Selecting backend ID`
			`Specify the selection for the BackendID`
			
			`EJB Module:  Increment Enterprise Java Bean`
			`URI:  Increment.jar,META-INF/ejb-jar.xml`
			`BackendId list:  CLOUDSCAPE_V50_1`
			`CurrentBackendId:  [CLOUDSCAPE_V50_1]:`
			
			`Task[21]: Specifying application options`
			`Specify the various options available`
			`to prepare and install your application.`
			
			`Pre-compile JSP:  [No]:`
			`Deploy EJBs:  [No]:`
			`Deploy WebServices:  [No]:`

updateInteractive continued		• partialapp - Indicates that you want to update a partial application. Using subset of application components provided in a zip file format you can update, add, and delete files and modules. The zip file is not a valid Java 2 platform, Enterprise Edition (J2EE) archive. Instead, this file contains application artifacts in the same hierarchical structure as they are displayed in an EAR file. For more information on how to construct the partial application zip file, see the Java API section. If you indicate the partialapp value as the content type, use the contents option to specify the location of the zip file. When a partial application is provided as an update input, the binding information and application options cannot be specified and are ignored, if provided. For a list of the valid options for the **updateInteractive** command, see "Options for the AdminApp object install, installInteractive, edit, editInteractive, update, and updateInteractive commands" on page 437.	Task[22]: Specifying EJB deploy options Specify the options to deploy EJB. ....EJB Deploy option is not enabled. Task[24]: Copy WSDL files Copy WSDL files ....This task does not require any user input Task[25]: Specify options to deploy Web services Specify options to deploy Web services ....Web Services deploy option is not enabled. Update of myApp has started. ADMA5009I: Application archive extracted at C:\DOCUME~1\lavena\LOCALS~1\Temp\ app_fb5a48e969\ext/Increment.jar FileMergeTask completed successfully ADMA5005I: Application myApp configured in WebSphere repository delFiles: [] delM: null addM: [Increment.jar,] Pattern for remove loose and mod: Loose add pattern: META-INF/[^/]*|WEB-INF/[^/]*|.*wsdl root file to be copied: META-INF/application.xml to C:\asv\b0403.04\WebSphere\AppServer\ wstemp\Scriptfb5a487089\ workspace\cells\BAMBIE\applications\testSM.ear\deployments\ testSM/META-INF/application.xml del files for full module add/update: [] ADMA6017I: Saved document C:\asv\b0403.04\WebSphere\AppServer\ wstemp\Scriptfb5a487089\workspace\cells\ BAMBIE\applications\ testSM.ear\deployments\testSM/Increment.jar\ META-INF/ejb-jar.xml ADMA6016I: Add to workspace Increment.jar/META-INF/ejb-jar.xml ADMA6017I: Saved document C:\asv\b0403.04\WebSphere\AppServer\ wstemp\Scriptfb5a487089\workspace\cells\ BAMBIE\applications\ testSM.ear\deployments\testSM/Increment.jar\ META-INF/MANIFEST.MF ADMA6016I: Add to workspace Increment.jar/ META-INF/MANIFEST.MF ADMA6017I: Saved document C:\asv\b0403.04\ WebSphere\AppServer\ wstemp\Scriptfb5a487089\workspace\cells\ BAMBIE\applications\ testSM.ear\deployments\testSM/Increment.jar\ META-INF/ibm-ejb-jar-bnd.xmi ADMA6016I: Add to workspace Increment.jar/ META-INF/ibm-ejb-jar-bnd.xmi ADMA6017I: Saved document C:\asv\b0403.04\ WebSphere\AppServer\ wstemp\Scriptfb5a487089\workspace\cells\ BAMBIE\applications\ testSM.ear\deployments\testSM/Increment.jar\ META-INF/Table.ddl ADMA6016I: Add to workspace Increment.jar/ META-INF/Table.ddl

updateInteractive continued			ADMA6017I: Saved document C:\asv\b0403.04\ WebSphere\ AppServer\wstemp\Scriptfb5a487089\workspace\ cells\BAMBIE\ applications\testSM.ear\deployments\testSM/ Increment.jar\META-INF/ibm-ejb-jar-ext.xmi ADMA6016I: Add to workspace Increment.jar/ META-INF/ibm-ejb-jar-ext.xmi add files for full module add/update: [Increment.jar/ META-INF/ejb-jar.xml, Increment.jar/ META-INF/MANIFEST.MF, Increment.jar/META-INF/ibm-ejb-jar-bnd.xmi, Increment.jar/META-INF/ Table.ddl, Increment.jar/META-INF/ibm-ejb-jar-ext.xmi] ADMA5005I: Application myApp configured in WebSphere repository xmlDoc: [#document: null] root element: [app-delta: null] ****** delta file name: C:\asv\b0403.04\ WebSphere\ AppServer\wstemp\Scriptfb5a487089\workspace \cells\BAMBIE\applications\testSM.ear/ deltas/delta-1079551520393 ADMA5005I: Application myApp configured in WebSphere repository ADMA6011I: Deleting directory tree C:\DOCUME~1\lavena\LOCALS~1\Temp\ app_fb5a48e969 ADMA5011I: Cleanup of temp dir for app myApp done. Update of myApp has ended.

Options for the AdminApp object install, installInteractive, edit, editInteractive, update, and updateInteractive commands: This article lists the available options for the **install**, **installInteractive**, **edit**, **editInteractive**, **update**, and **updateInteractive** commands of the AdminApp object. The options listed in this article apply to all of these commands except where noted.

See Commands for the AdminApp object for more detailed information on how to use the commands. See Usage table for the options of the AdminApp object install, installInteractive, update, updateInteractive, edit, and editInteractive commands for a list of applicable commands for each option.

The following options are available for the **install**, **installInteractive**, **edit**, **editInteractive**, **update**, and **updateInteractive** commands:

Option name:	Description:	Examples:

ActSpecJNDI	Binds J2C activation specs to destination JNDI names. You can bind J2C activation specs in your application or module to a destination JNDI name. This option is optional. Each element of the ActSpecJNDI option consists of the following fields: RARModule, uri, j2cid, j2c.jndiName. j2c.jndiName field, can be assigned a value. The current contents of the option after running default bindings include:
	• RARModule: <rar module name>
	• uri: <rar name>,META-INF/ra.xml
	• j2cid: <messageListenerType>
	• j2c.jndiName: null
	• Object identifier: javax.jms.MessageListener
	You can only use this option if the activation spec has the `Destination` property defined in the ra.xml file and the introspected type of the `Destination` property is the following: `javax.jms.Destination`
		Using Jacl:
		`$AdminApp install $embeddedEar {-ActSpecJNDI {{"FVT Resource Adapter" jca15cmd.rar,META-INF/ra.xml javax.jms.MessageListener jndi5} {"FVT Resource Adapter" jca15cmd.rar,META-INF/ra.xml javax.jms.MessageListener2 jndi6}}}`
		Using Jython:
		`AdminApp.install(embeddedEar, ['-ActSpecJNDI', [["FVT Resource Adapter", 'jca15cmd.rar,META-INF/ra.xml', 'javax.jms.MessageListener', 'jndi5'], ["FVT Resource Adapter", 'jca15cmd.rar,META-INF/ra.xml', 'javax.jms.MessageListener2', 'jndi6']]])`

ActSpecJNDI continued	Use the **taskInfo** command of the AdminApp object to obtain information about the data that is needed for your application. You need to provide data for rows or entries that are either missing information, or require an update.	
allowPermInFilter Policy	Specifies to continue with the application deployment process even when the application contains policy permissions that are in the filter policy. This option does not require a value.	
appname	Specifies the name of the application. The default is the display name of the application.	

BackendIdSelection	Specifies the backend ID for the enterprise bean Java archive (JAR) modules that have container-managed persistence (CMP) beans. An enterprise bean JAR module can support multiple backend configurations as specified using an application assembly tool.
	Use this option to change the backend ID during installation.
		Using Jacl:
		`$AdminApp install c:/myapp.ear {-BackendIdSelection`
		`{{Annuity20EJB Annuity20EJB.jar,META-INF/ejb-jar.xml  DB2UDBNT_V72_1}}}`
		Using Jython:
		`AdminApp.install('c:/myapp.ear', '[-BackendIdSelection`
		`[[Annuity20EJB Annuity20EJB.jar,META-INF/ejb-jar.xml  DB2UDBNT_V72_1]]]')`

BindJndiForEJB MessageBinding	Binds enterprise beans to listener port names or Java Naming and Directory Interface (JNDI) names. Use this option to provide missing data or update a task. Ensure each message-driven enterprise bean in your application or module is bound to a listener port name.
	Each element of the BindJndiForEJBMessageBinding option consists of the following fields: EJBModule, EJB, uri, listenerPort, JNDI, jndi.dest, and actspec.auth. Some of these fields, can be assigned values: listenerPort, JNDI, jndi.dest, and actspec.auth.
	The current contents of the option after running default bindings include:
	• EJBModule: Ejb1
	• EJB: MessageBean
	• uri: ejb-jar-ic.jar,META-INF/ejb-jar.xml
	• listenerPort: MessageBeanPort
	• JNDI: null
	• jndi.dest: null
	• actspec.auth: null
		Using Jacl:
		`$AdminApp install $ear {-BindJndiForEJBMessageBinding`
		`{{Ejb1 MessageBean`
		`ejb-jar-ic.jar,META-INF/ejb-jar.xml myListenerPort`
		`jndi1 jndiDest1 actSpecAuth1}}}`
		Using Jython:
		`AdminApp.install(ear, ['-BindJndiForEJBMessageBinding',`
		`[['Ejb1', 'MessageBean',`
		`'ejb-jar-ic.jar,META-INF/ejb-jar.xml', 'myListenerPort',`
		`'jndi1', 'jndiDest1', 'actSpecAuth1']]])`

BindJndiForEJB MessageBinding continued	Use the **taskInfo** command of the AdminApp object to obtain information about the data that is needed for your application. You need to provide data for rows or entries that are either missing information, or require an update.	

BindJndiForEJB NonMessageBinding	Binds enterprise beans to Java Naming and Directory Interface (JNDI) names.
	Ensure each non message-driven enterprise bean in your application or module is bound to a JNDI name. Use this option to provide missing data or update a task.
	Use the **taskInfo** command of the AdminApp object to obtain information about the data that is needed for your application. You need to provide data for rows or entries that are either missing information, or require an update.
		Using Jacl:
		`$AdminApp install c:/myapp.ear {-BindJndiForEJBNonMessageBinding {{"Increment Bean Jar" Inc Increment.jar,META-INF/ejb-jar.xml IncBean}}}`
		Using Jython:
		`AdminApp.install('c:/myapp.ear', '[-BindJndiForEJBNonMessageBinding [["Increment Bean Jar" Inc Increment.jar,META-INF/ejb-jar.xml IncBean]]]')`

cell	Specifies the cell name to install or update an entire application or to update an application in order to add a new module. If you want to update an entire application, this option only applies if the application contains a new module that does not exist in the installed application.	
cluster	Specifies the cluster name to install or update an entire application or to update an application in order to add a new module. This option only applies in a Network Deployment environment. If you want to update an entire application, this option only applies if the application contains a new module that does not exist in the installed application.	

contents	Specifies the file that contains the content that you want to update. For example, depending on the content type, the file could be an EAR file, a module, a partial zip, or a single file. The path to the file must be local to the scripting client. The `contents` option is required unless you have specified the `delete` option.	
contenturi	Specifies the URI of the file that you are adding, updating, or removing from an application. This option only applies to the **update** command. The `contenturi` option is required if the content type is `file` or `modulefile`. This option is ignored for other content types.	
contextroot	Specifies the context root that you use when installing a stand-alone Web archive (WAR) file.	

CorrectOracleIsolation Level	Specifies the isolation level for the Oracle type provider. Use this option to provide missing data or to update a task.
	The last field of each entry specifies the isolation level. Valid isolation level values are `2` or `4`.
	Use the **taskInfo** command of the AdminApp object to obtain information about the data that is needed for your application. You only need to provide data for rows or entries that are either missing information, or require an update.
		Using Jacl:
		`$AdminApp install c:/myapp.ear {-CorrectOracleIsolationLevel {{AsyncSender jms/MyQueueConnectionFactory jms/Resource1 2}}`
		Using Jython:
		`AdminApp.install('c:/myapp.ear', '[-CorrectOracleIsolationLevel [[AsyncSender jms/MyQueueConnectionFactory jms/Resource1 2]]]')`

CorrectUseSystem Identity	Replaces RunAs System to RunAs Roles.
	The enterprise beans that you install contain a RunAs system identity. You can optionally change this identity to a RunAs role. Use this option to provide missing data or update a task.
	Use the **taskInfo** command of the AdminApp object to obtain information about the data that is needed for your application. You need to provide data for rows or entries that are either missing information, or require an update.
		Using Jacl:
		`$AdminApp install c:/myapp.ear {-CorrectUseSystemIdentity`
		` {{Inc "Increment Bean Jar"`
		`Increment.jar,META-INF/ejb-jar.xml getValue() RunAsUser2`
		` user2 password2} {Inc "Increment`
		`Bean Jar" Increment.jar,META-INF/ejb-jar.xml Increment()`
		` RunAsUser2 user2 password2}}}`
		Using Jython:
		`AdminApp.install('c:/myapp.ear', '[-CorrectUseSystemIdentity`
		` [[Inc "Increment Bean Jar"`
		`Increment.jar,META-INF/ejb-jar.xml getValue() RunAsUser2`
		` user2 password2] [Inc "Increment`
		`Bean Jar" Increment.jar,META-INF/ejb-jar.xml Increment()`
		`RunAsUser2 user2 password2]]]')`

createMBeansFor Resources	Specifies that MBeans are created for all resources such as, servlets, JavaServer Pages (JSP) files, and enterprise beans, that are defined in an application when the application starts on a deployment target. This option does not require a value. The default setting is the nocreateMBeansForResources option.	

custom	Specifies a name-value pair using the format `name=value`. Use the `custom` option to pass options to application deployment extensions. See the application deployment extension documentation for available custom options.	

DataSourceFor10 CMPBeans	Specifies optional data sources for individual 1.x container-managed persistence (CMP) beans. Use this option to provide missing data or to update a task.
	Mapping a specific data source to a CMP bean overrides the default data source for the module that contains the enterprise bean. Each element of the DataSourceFor10CMPBeans option consists of the following fields: EJBModule, EJB, uri, JNDI, userName, password, login.config.name, and auth.props. Of these fields, the following can be assigned values: JNDI, userName, password, login.config.name, and auth.props.
		Using Jacl:
		`$AdminApp install c:/app1.ear {-DataSourceFor10CMPBeans`
		`  {{"Increment CMP 1.1 EJB"`
		`IncCMP11 IncCMP11.jar,META-INF/ejb-jar.xml myJNDI user1 password1`
		`loginName1 authProps1}}}`
		Using Jython:
		`AdminApp.install('c:/app1.ear', ['-DataSourceFor10CMPBeans',`
		` [["Increment CMP 1.1 EJB", 'IncCMP11',`
		` 'IncCMP11.jar,META-INF/ejb-jar.xml', 'myJNDI', 'user1',`
		` 'password1', 'loginName1', 'authProps1']]])`

DataSourceFor10 CMPBeans continued	The current contents of the option after running default bindings include: - EJBModule: Increment CMP 1.1 EJB - EJB: IncCMP11 - uri: IncCMP11.jar,META-INF/ejb-jar.xml - JNDI: DefaultDatasource - userName: null - password: null - login.config.name: DefaultPrincipalMapping - auth.props: - LoginConfiguration: Name - Properties If the login.config.name is set to DefaultPrincipalMapping, a property is created with the name com.ibm.mapping.authDataAlias . The value of the property is set by the auth.props. If the login.config name is not set to DefaultPrincipalMapping, the auth.props can specify multiple properties. The string format is websphere:name= <name1>,value=<value1>,description=<desc1>. Specify multiple properties using the plus sign (+) .	

DataSourceFor10 CMPBeans continued	Use the **taskInfo** command of the AdminApp object to obtain information about the data that is needed for your application. You need to provide data for rows or entries that are missing information, or require an update.	

DataSourceFor20 CMPBeans	Specifies optional data sources for individual 2.x container-managed persistence (CMP) beans. Use this option to provide missing data or to update a task.
	Mapping a specific data source to a CMP bean overrides the default data source for the module that contains the enterprise bean. Each element of the DataSourceFor20CMPBeans option consists of the following fields: EJBModule, EJB, uri, JNDI, resAuth, login.config.name, and auth.props. Of these fields, the following can be assigned values: JNDI, resAuth, login.config.name, and auth.props.
		Using Jacl:
		`$AdminApp install c:/app1.ear {-DataSourceFor20CMPBeans {{"Increment Enterprise Java Bean" Increment Increment.jar, META-INF/ejb-jar.xml jndi1 container}}}`
		Using Jython:
		`AdminApp.install('c:/app1.ear', ['-DataSourceFor20CMPBeans', [["Increment Enterprise Java Bean", 'Increment', 'Increment.jar, META-INF/ejb-jar.xml', 'jndi1', 'container']]])`

DataSourceFor20 CMPBeans continued	The current contents of the option after running default bindings includes the following: • EJBModule: Increment enterprise bean • EJB: Increment • uri: Increment.jar,META-INF/ejb-jar.xml • JNDI: DefaultDatasource • resAuth: cmpBinding.perConnectionFactory • login.config.name: DefaultPrincipalMapping • auth.props: • LoginConfiguration: Name • Properties	

DataSourceFor20 CMPBeans continued	The last field in each entry of this task specifies the value for resource authorization. Valid values for resource authorization are `per connection factory` or `container`. If the login.config.name is set to DefaultPrincipalMapping, a property is created with the name com.ibm.mapping.authDataAlias . The value of the property is set by the auth.props. If the login.config name is not set to DefaultPrincipalMapping, the auth.props can specify multiple properties. The string format is websphere:name= <name1>,value=<value1>,description=<desc1>. Specify multiple properties using the plus sign (+) . Use the **taskInfo** command of the AdminApp object to obtain information about the data needed for your application. You only need to provide data for rows or entries that are missing information, or require an update.	

DataSourceFor10 EJBModules	Specifies the default data source for the enterprise bean module that contains 1.x container-managed persistence (CMP) beans. Use this option to provide missing data or update a task.
	Each element of the DataSourceFor10EJBModules option consists of the following fields: EJBModule, uri, JNDI, userName, password, login.config.name, and auth.props. Of these fields, the following can be assigned values: JNDI, userName, password, login.config.name, and auth.props.
		Using Jacl:
		`$AdminApp install c:/app1.ear {-DataSourceFor10EJBModules {{"Increment CMP 1.1 EJB" IncCMP11.jar,META-INF/ejb-jar.xml yourJNDI user2 password2 loginName authProps}}}`
		Using Jython:
		`AdminApp.install('c:/app1.ear', ['-DataSourceFor10EJBModules', [["Increment CMP 1.1 EJB", 'IncCMP11.jar,META-INF/ejb-jar.xml', 'yourJNDI', 'user2', 'password2', 'loginName', 'authProps']]])`

DataSourceFor10 EJBModules continued	The current contents of the option after running default bindings include: • EJBModule: Increment CMP 1.1 enterprise bean • uri: IncCMP11.jar,META-INF/ejb-jar.xml • JNDI: DefaultDatasource • userName: null • password: null • login.config.name: DefaultPrincipalMapping • auth.props: • LoginConfiguration: Name • Properties If the login.config.name is set to DefaultPrincipalMapping, a property is created with the name com.ibm.mapping.authDataAlias . The value of the property is set by the auth.props. If the login.config name is not set to DefaultPrincipalMapping, the auth.props can specify multiple properties. The string format is websphere:name= <name1>,value=<value1>,description=<desc1>. Specify multiple properties using the plus sign (+) .	

DataSourceFor10 EJBModules continued	Use the **taskInfo** command of the AdminApp object to obtain information about the data that is needed for your application. You need to provide data for rows or entries that are either missing information, or require an update.	

DataSourceFor20 EJBModules	Specifies the default data source for the enterprise bean 2.x module that contains 2.x container managed persistence (CMP) beans. Use this option to provide missing data or update a task.
	Each element of the DataSourceFor20EJBModules option consists of the following fields: EJBModule, uri, JNDI, resAuth, login.config.name, and auth.props. Of these fields, the following can be assigned values: JNDI, resAuth, login.config.name, and auth.props.
	The current contents of the option after running default bindings include:
	• EJBModule: Increment enterprise bean
	• uri: Increment.jar,META-INF/ejb-jar.xml
	• JNDI: DefaultDatasource
	• resAuth: cmpBinding.perConnectionFactory
	• login.config.name: DefaultPrincipalMapping
	• auth.props:
	• LoginConfiguration: Name
	• Properties
		Using Jacl:
		`$AdminApp install c:/app1.ear {-DataSourceFor20EJBModules`
		`{{"Increment Enterprise Java Bean" Increment.jar,META-INF/ejb-jar.xml`
		` jndi2 container}}}`
		Using Jython:
		`AdminApp.install('c:/app1.ear', ['-DataSourceFor20EJBModules',`
		`[["Increment Enterprise Java Bean", 'Increment.jar,META-INF/ejb-jar.xml',`
		` 'jndi2', 'container']]])`

DataSourceFor20 EJBModules	The last field in each entry of this task specifies the value for resource authorization. Valid values for resource authorization are `per connection factory` or `container`. If the login.config.name is set to DefaultPrincipalMapping, a property is created with the name com.ibm.mapping.authDataAlias . The value of the property is set by the auth.props. If the login.config name is not set to DefaultPrincipalMapping, the auth.props can specify multiple properties. The string format is websphere:name= <name1>,value=<value1>,description=<desc1>. Specify multiple properties using the plus sign (+) . Use the **taskInfo** command of the AdminApp object to obtain information about the data that is needed for your application. You need to provide data for rows or entries that are either missing information, or require update.	

defaultbinding.cf.jndi	Specifies the Java Naming and Directory Interface (JNDI) name for the default connection factory.	
defaultbinding.cf.resauth	Specifies the RESAUTH for the connection factory.	
defaultbinding.datasource.jndi	Specifies the Java Naming and Directory Interface (JNDI) name for the default data source.	
defaultbinding.datasource.password	Specifies the password for the default data source.	
defaultbinding.datasource.username	Specifies the user name for the default data source.	
defaultbinding.ejbjndi.prefix	Specifies the prefix for the enterprise bean Java Naming and Directory Interface (JNDI) name.	
defaultbinding.force	Specifies that the default bindings override the current bindings.	
defaultbinding.strategy.file	Specifies a custom default bindings strategy file.	
defaultbinding.virtual.host	Specifies the default name for a virtual host.	
depl.extension.reg	Deprecated. No replication option is available.	

deployejb	Specifies to run the EJBDeploy tool during installation. This option does not require a value. If you pre-deploy the application Enterprise Archive (EAR) file using the EJBDeploy tool then the default value is `nodeployejb`. If not, the default value is `deployejb`.	
deployejb.classpath	Specifies an extra class path for the EJBDeploy tool.	
deployejb.dbschema	Specifies the database schema for the EJBDeploy tool.	

deployejb.dbtype	Specifies the database type for the EJBDeploy tool.
	Possible values include:
	`CLOUDSCAPE_V5`
	`DB2UDB_V72`
	`DB2UDBOS390_V6`
	`DB2UDBISERIES`
	`INFORMIX_V73`
	`INFORMIX_V93`
	`MSSQLSERVER_V7`
	`MSSQLSERVER_2000`
	`ORACLE_V8`
	`ORACLE_V9I`
	`SYBASE_V1200`
	For a list of current supported database vendor types, run `ejbdeploy -?`.	

deployejb.rmic	Specifies extra RMIC options to use for the EJBDeploy tool.	

deployws	Specifies to deploy Web services during installation. This option does not require a value. The default value is: `nodeployws`.	
deployws.classpath	Specifies the extra class path to use when you deploy Web services.	
deployws.jardirs	Specifies the extra extension directories to use when you deploy Web services.	
distributeApp	Specifies that the application management component distributes application binaries. This option does not require a value. This setting is the default.	

EmbeddedRar	Binds Java 2 Connector objects to JNDI names. You must bind each Java 2 Connector object in your application or module, such as, J2C connection factories, J2C activation specs and J2C administrative objects, to a JNDI name. Each element of the EmbeddedRar option contains the following fields: RARModule, uri, j2cid, j2c.name, j2c.jndiName. You can assign the following values to the fields: j2c.name, j2c.jndiName. The current contents of the option after running default bindings include: RARModule: <rar module name> uri: <rar name>,META-INF/ra.xml j2cid: <identifier of the J2C object> j2c.name: j2cid j2c.jndiName: eis/j2cid Where j2cid is: J2C connection factory :connectionFactoryInterface J2C admin object :adminObjectInterface J2C activation spec : message listener type If the ID is not unique in the ra.xml file, -<number> will be added. For example, javax.sql.DataSource-2.	Using Jacl: $AdminApp install $embeddedEar {-EmbeddedRar {{"FVT Resource Adapter" jca15cmd.rar,META-INF/ra.xml javax.sql.DataSource javax.sql.DataSource1 eis/javax.sql.javax.sql.DataSSource1} {"FVT Resource Adapter" jca15cmd.rar,META-INF/ra.xml javax.sql.DataSource2 javax.sql.DataSource2 eis/javax.sql.DataSource2} {"FVT Resource Adapter" jca15cmd.rar,META-INF/ra.xml javax.jms.MessageListener javax.jms.MessageListener1 eis/javax.jms.MessageListener1} {"FVT Resource Adapter" jca15cmd.rar,META-INF/ra.xml javax.jms.MessageLListener2 javax.jms.MessageListener2 eis/javax.jms.MessageListener2} {"FVT Resource Adapter" jca15cmd.rar,META-INF/ra.xml fvt.adapter.message.FVTMessageProvider fvt.adapter.message. FVTMessageProvider1 eis/fvt.adapter.message.FVTMessageProvider1} {"FVT Resource Adapter" jca15cmd.rar,META-INF/ra.xml fvt.adapter.message.FVTMessageProvider2 fvt. adapter.message.FVTMessageProvider2 eis/fvt.adapter. message.FVTMessageProvider2}}} Using Jython: AdminApp.install(embeddedEar, ['-EmbeddedRar', [["FVT Resource Adapter", 'jca15cmd.rar,META-INF/ra.xml', 'javax.sql.DataSource', 'javax.sql.DataSource1', 'eis/javax.sql.javax.sql.DataSSource1'], ["FVT Resource Adapter", 'jca15cmd.rar,META-INF/ra.xml javax.sql.DataSource2', 'javax.sql.DataSource2', 'eis/javax.sql.DataSource2'], ["FVT Resource Adapter", 'jca15cmd.rar,META-INF/ra.xml', 'javax.jms.MessageListener', 'javax.jms.MessageListener1', 'eis/javax.jms.MessageListener1'], ["FVT Resource Adapter", 'jca15cmd.rar,META-INF/ra.xml', 'javax.jms.MessageLListener2', 'javax.jms.MessageListener2', 'eis/javax.jms.MessageListener2'], ["FVT Resource Adapter", 'jca15cmd.rar,META-INF/ra.xml fvt.adapter.message.FVTMessageProvider', 'fvt.adapter.message.FVTMessageProvider1', 'eis/fvt.adapter.message. FVTMessageProvider1'], ["FVT Resource Adapter", 'jca15cmd.rar,META-INF/ ra.xml', 'fvt.adapter.message.FVTMessageProvider2', 'fvt.adapter.message. FVTMessageProvider2', 'eis/fvt.adapter.message.FVTMessageProvider2']]])

EmbeddedRar continued	Use the **taskInfo** command of the AdminApp object to obtain information about the data that is needed for your application. You need to provide data for rows or entries that are either missing information, or require an update.	

EnsureMethod ProtectionFor10EJB	Selects method protections for unprotected methods of 1.x enterprise beans. Specify to leave the method as unprotected, or assign protection which denies all access. Use this option to provide missing data or to update a task.
	Use the **taskInfo** command of the AdminApp object to obtain information about the data that is needed for your application. You need to provide data for rows or entries that are either missing information, or require an update.
		Using Jacl:
		`$AdminApp install c:/myapp.ear {-EnsureMethodProtectionFor10EJB`
		` {{"Increment EJB Module"`
		` IncrementEJBBean.jar,META-INF/ejb-jar.xml ""} {"Timeout EJB Module"`
		` TimeoutEJBBean.jar,META-INF/ejb-jar.xml`
		` methodProtection.denyAllPermission}}}`
		Using Jython:
		`AdminApp.install('c:/myapp.ear', '[-EnsureMethodProtectionFor10EJB`
		` [["Increment EJB Module"`
		`IncrementEJBBean.jar,META-INF/ejb-jar.xml ""] ["Timeout EJB Module"`
		` TimeoutEJBBean.jar,META-INF/ejb-jar.xml`
		` methodProtection.denyAllPermission]]]')`
		The last field in each entry of this task specifies the value of the protection. Valid protection values include: `methodProtection.denyAllPermission`. You can also leave the value blank if you want the method to remain unprotected.

EnsureMethodProtectionFor20EJB	Selects method protections for unprotected methods of 2.x enterprise beans. Specify to assign a security role to the unprotected method, add the method to the exclude list, or mark the method as cleared. You can assign multiple roles for a method by separating roles names with commas. Use this option to provide missing data or to update a task.
	Use the **taskInfo** command of the AdminApp object to obtain information about the data that is needed for your application. You need to provide data for rows or entries that are either missing information, or require an update the existing data.
		Using Jacl:
		`$AdminApp install c:/myapp.ear {-EnsureMethodProtectionFor20EJB`
		` {{CustomerEjbJar`
		`customerEjb.jar,META-INF/ejb-jar.xml methodProtection.uncheck}`
		` {SupplierEjbJar`
		`supplierEjb.jar,META-INF/ejb-jar.xml methodProtection.exclude}}}`
		Using Jython:
		`AdminApp.install('c:/myapp.ear', '[-EnsureMethodProtectionFor20EJB`
		` [[CustmerEjbJar`
		`customerEjb.jar,META-INF/ejb-jar.xml methodProtection.uncheck]`
		` [SupplierEjbJar`
		`supplierEjb.jar,META-INF/ejb-jar.xml methodProtection.exclude]]]')`
		The last field in each entry of this task specifies the value of the protection. Valid protection values include: `methodProtection.uncheck`, `methodProtection.exclude`, or a list of security roles that are separated by commas.

installdir	Deprecated. This option is replaced by the `installed.ear.destination` option.	
installed.ear.destination	Specifies the directory to place application binaries.	

MapModulesToServers	Specifies the application server where you want to install modules that are contained in your application. You can install modules on the same server, or disperse them among several servers. Use this option to provide missing data or to update to a task.
	
	Use the **taskInfo** command of the AdminApp object to obtain information about the data that is needed for your application. You need to provide data for rows or entries that are either missing information, or require an update.
		Using Jacl:
		`$AdminApp install c:/myapp.ear {-MapModulesToServers`
		` {{"Increment Bean Jar"`
		`Increment.jar,META-INF/ejb-jar.xml WebSphere:cell=mycell,`
		` node=mynode,server=server1}`
		`{"Default Application" default_app.war,WEB-INF/web.xml WebSphere:`
		`cell=mycell,node=mynode,server=server1}`
		`{"Examples Application" examples.war,WEB-INF/web.xml WebSphere:`
		`cell=mycell,node=mynode,server=server1}}}`
		
		Using Jython:
		`AdminApp.install('c:/myapp.ear', '[-MapModulesToServers`
		` [["Increment Bean Jar"`
		`Increment.jar,META-INF/ejb-jar.xml WebSphere:cell=mycell,`
		`node=mynode,server=server1]`
		`["Default Application" default_app.war,WEB-INF/web.xml`
		`WebSphere:cell=mycell,node=mynode,server=server1]`
		`["Examples Application" examples.war,WEB-INF/web.xml WebSphere:`
		`cell=mycell,node=mynode,server=server1]]]')`

MapEJBRefToEJB	Maps enterprise Java references to enterprise beans. You must map each enterprise bean reference defined in your application to an enterprise bean. Use this option to provide missing data or update to a task.
	Use the **taskInfo** command of the AdminApp object to obtain information about the data needed for your application. You only need to provide data for rows or entries that are missing information, or those where you want to update the existing data.
		Using Jacl:
		`$AdminApp install c:/myapp.ear {-MapEJBRefToEJB`
		` {{"Examples Application" ""`
		`examples.war,WEB-INF/web.xml BeenThereBean com.ibm.websphere.`
		`beenthere.BeenThere IncBean}}}`
		Using Jython:
		`AdminApp.install('c:/myapp.ear', '[-MapEJBRefToEJB`
		` [["Examples Application" ""`
		`examples.war,WEB-INF/web.xml BeenThereBean com.ibm.websphere.`
		`beenthere.BeenThere IncBean]]]')`

MapMessage DestinationRefToEJB	Maps message destination references to Java Naming and Directory Interface (JNDI) names of administrative objects from the installed resource adapters. You must map each message destination reference that is defined in your application to an administrative object. Use this option to provide missing data or to update a task.
	The current contents of the option after running default bindings include:
	• EJB Module: ejb-jar-ic.jar
	• EJB: MessageBean
	• URI: ejb-jar-ic.jar,META-INF/ejb-jar.xml
	• JNDI Name: [eis/J2CACT1]:
	• Destination JNDI Name: [jms/TopicName]:
	• The default JNDI name will be picked up from the corresponding message destination reference.
		Using Jacl:
		`$AdminApp install $earfile {-MapMessageDestinationRefToEJB`
		`{{ejb-jar-ic.jar Publisher ejb-jar-ic.jar,META-INF/ejb-jar.xml`
		` MyConnection jndi2}`
		`{ejb-jar-ic.jar Publisher ejb-jar-ic.jar,META-INF/ejb-jar.xml`
		` PhysicalTopic jndi3}`
		`{ejb-jar-ic.jar Publisher ejb-jar-ic.jar,META-INF/ejb-jar.xml`
		` jms/ABC jndi4}}}`
		Using Jython:
		`AdminApp.install(ear1, ['-MapMessageDestinationRefToEJB',`
		`[['ejb-jar-ic.jar', 'Publisher', 'ejb-jar-ic.jar,META-INF/`
		`ejb-jar.xml', 'MyConnection', 'jndi2'],`
		`['ejb-jar-ic.jar', 'Publisher', 'ejb-jar-ic.jar,META-INF/`
		`ejb-jar.xml', 'PhysicalTopic', 'jndi3'],`
		`['ejb-jar-ic.jar', 'Publisher', 'ejb-jar-ic.jar,META-INF/`
		`ejb-jar.xml', 'jms/ABC', 'jndi4']]])`

MapMessage DestinationRefToEJB continued	Use the **taskInfo** command of the AdminApp object to obtain information about the data that is needed for your application. You need to provide data for rows or entries that are either missing information, or require an update.	

MapResEnvRefToRes	Maps resource environment references to resources. You must map each resource environment reference that is defined in your application to a resource. Use this option to provide missing data or to update a task.
	Use the **taskInfo** command of the AdminApp object to obtain information about the data that is needed for your application. You need to provide data for rows or entries that are either missing information, or require an update.
		Using Jacl:
		`$AdminApp install c:/myapp.ear {-MapResEnvRefToRes`
		`{{AsyncSender AsyncSender`
		`asyncSenderEjb.jar,META-INF/ejb-jar.xml jms/ASYNC_SENDER_QUEUE`
		`javax.jms.Queue jms/Resource2}}}`
		Using Jython:
		`AdminApp.install('c:/myapp.ear', '[-MapResEnvRefToRes`
		`[[AsyncSender AsyncSender`
		`asyncSenderEjb.jar,META-INF/ejb-jar.xml jms/ASYNC_SENDER_QUEUE`
		`javax.jms.Queue jms/Resource2]]]')`

MapResRefToEJB	Maps resource references to resources. You must map each resource reference that is defined in your application to a resource. Use this option to provide missing data or to update a task.
	Each element of the MapResRefToEJB option consists of the following fields: module, EJB, uri, referenceBinding, resRef.type, JNDI, login.config.name, and auth.props. Of these fields, the following can be assigned values: JNDI, login.config.name, auth.props. The JNDI field is required.
	The current contents of the option after running default bindings include:
	• Module: deplmtest.jar
	• EJB: MailEJBObject
	• uri: deplmtest.jar,META-INF/ejb-jar.xml
	• referenceBinding: mail/MailSession9
	• resRef.type: javax.mail.Session
	• JNDI: mail/DefaultMailSession
	• login.config.name: DefaultPrincipalMapping
	• auth.props:
		Using Jacl:
		`$AdminApp install c:/app1.ear {-MapResRefToEJB {{deplmtest.jar`
		`MailEJBObject deplmtest.jar,META-INF/ejb-jar.xml mail/MailSession9`
		`javax.mail.Session jndi1 login1 authProps1} {"JavaMail Sample WebApp" ""`
		`mtcomps.war,WEB-INF/web.xml mail/MailSession9 javax.mail.`
		`Session jndi2 login2 authProps2}}}`
		Using Jython:
		`AdminApp.install('c:/app1.ear', ['-MapResRefToEJB', [['deplmtest.jar',`
		`'MailEJBObject', 'deplmtest.jar,META-INF/ejb-jar.xml mail/MailSession9',`
		`'javax.mail.Session', 'jndi1', 'login1', 'authProps1'],`
		`["JavaMail Sample WebApp", "",`
		`'mtcomps.war,WEB-INF/web.xml', 'mail/MailSession9', 'javax.mail.`
		`Session', 'jndi2', 'login2', 'authProps2']]])`

MapResRefToEJB continued	If the login.config.name is set to DefaultPrincipalMapping, a property is created with the name com.ibm.mapping.authDataAlias . The value of the property is set by the auth.props. If the login.config name is not set to DefaultPrincipalMapping, the auth.props can specify multiple properties. The string format is websphere:name= <name1>,value=<value1>,description=<desc1>. Specify multiple properties using the plus sign (+) . Use the **taskInfo** command of the AdminApp object to obtain information about the data that is needed for your application. You need to provide data for rows or entries that are either missing information, or require an update.	

MapRolesToUsers	Maps users to roles. You must map each role that is defined in the application or module to a user or group from the domain user registry. You can specify multiple users or groups for a single role by separating them with a pipe (|). Use this option to provide missing data or to update a task.
	Use the **taskInfo** command of the AdminApp object to obtain information about the data that is needed for your application. You need to provide data for rows or entries that are either missing information, or require an update.
		Using Jacl:
		`$AdminApp install c:/myapp.ear {-MapRolesToUsers {{"All Role" No Yes "" ""} {"Every Role" Yes No "" ""} {DenyAllRole No No user1 group1}}}`
		Using Jython:
		`AdminApp.install('c:/myapp.ear', '[-MapRolesToUsers [["All Role" No Yes "" ""] ["Every Role" Yes No "" ""] [DenyAllRole No No user1 group1]]]')`
		where `{{″All Role″ No Yes ″″ ″″}` corresponds to the following:
		″**All Role**″
		Represents the role name
		No Indicates to allow access to everyone (yes/no)
		Yes Indicates to allow access to all authenticated users (yes/no)
		″″ Indicates the mapped users
		″″ Indicates the mapped groups

MapRunAsRolesToUsers	Maps RunAs Roles to users. The enterprise beans you that install contain predefined RunAs roles. Enterprise beans that need to run as a particular role for recognition while interacting with another enterprise bean use RunAs roles. Use this option to provide missing data or to update a task.
	Use the **taskInfo** command of the AdminApp object to obtain information about the data that is needed for your application. You need to provide data for rows or entries that are either missing information, or require an update.
		Using Jacl:
		`$AdminApp install c:/myapp.ear {-MapRunAsRolesToUsers`
		`{{UserRole user1 password1}`
		`{AdminRole administrator administrator}}}`
		Using Jython:
		`AdminApp.install('c:/myapp.ear', '[-MapRunAsRolesToUsers`
		`[[UserRole user1 password1]`
		`[AdminRole administrator administrator]]]')`

MapWebModToVH	Selects virtual hosts for Web modules. Specify the virtual host where you want to install the Web modules that are contained in your application. You can install Web modules on the same virtual host, or disperse them among several hosts. Use this option to provide missing data or to update a task.
	Use the **taskInfo** command of the AdminApp object to obtain information about the data that is needed for your application. You need to provide data for rows or entries that are either missing information, or require an update.
		Using Jacl:
		`$AdminApp install c:/myapp.ear {-MapWebModToVH {{"Default Application"`
		`default_app.war,WEB-INF/web.xml default_host} {"Examples Application"`
		`examples.war,WEB-INF/web.xml default_host}}}`
		Using Jython:
		`AdminApp.install('c:/myapp.ear', '[-MapWebModToVH [["Default Application"`
		`default_app.war,WEB-INF/web.xml default_host] ["Examples Application"`
		`examples.war,WEB-INF/web.xml default_host]]]')`

noallowPermInFilterPolicy	Specifies not to continue with the application deployment process when the application contains policy permissions that are in the filter policy. This option is the default setting and it does not require a value.	

node	Specifies the node name to install or update an entire application or to update an application in order to add a new module. If you want to update an entire application, this option only applies if the application contains a new module that does not exist in the installed application.	
nocreateMBeansForResources	Specifies that MBeans are not created for all resources such as, servlets, JSPs, and enterprise beans, that are defined in an application when the application starts on a deployment target. This option is the default setting and it does not require a value.	
nodeployejb	Specifies not to run the EJBDeploy tool during installation. This option is the default setting and it does not require a value.	
nodeployws	Specifies not to deploy Web services during installation. This option is the default setting and it does not require a value.	

nodistributeApp	Specifies that the application management component does not distribute application binaries. This option does not require a value. The default setting is the distributeApp option.	
noreloadEnabled	Disables class reloading. This option does not require a value. The default setting is the `reloadEnabled` option.	
nopreCompileJSPs	Specifies not to precompile JavaServer Pages files. This option is the default setting and it does not require a value.	
noprocessEmbeddedConfig	Use this option to ignore the embedded configuration data that is include in the application. This option does not required a value. If the application Enterprise Archive (EAR) file does not contain embedded configuration data, the noprocessEmbeddedConfig option is the default setting. Otherwise, the default setting is the processEmbeddedConfig option.	

nouseMetaDataFromBinary	Specifies that the metadata that is used at run time, for example, deployment descriptors, bindings, extensions, and so on, come from the configuration repository. This option is the default setting and it does not require a value. Use this option to indicate that the metadata that is used at run time comes from the enterprise archive file (EAR) file.	
nousedefaultbindings	Specifies not to use default bindings for installation. This option is the default setting and it does not require a value.	

operation	Specifies the operation that you want to perform. This option only applies to the **update** command. The valid values include:
	- `add` - Adds new content.
	- `addupdate` - Adds or updates content based on the existence of content in the application.
	- `delete` - Deletes content.
	- `update` - Updates existing content.
	The operation option is required if the content type is file or modulefile. If the value of the content type is `app`, the value of the operation option must be `update`.
		The following examples show how to use the options for the **update** command to update a single file in a deployed application:
		Using Jacl:
		`$AdminApp update app1 file {-operation update -contents c:/apps/app1/my.xml -contenturi app1.jar/my.xml}`
		Using Jython string:
		`AdminApp.update('app1', 'file', '[-operation update -contents c:/apps/app1/my.xml -contenturi app1.jar/my.xml]')`
		Using Jython list:
		`AdminApp.update('app1', 'file', ['-operation', 'update', '-contents', 'c:/apps/app1/my.xml', '-contenturi', app1.jar/my.xml'])`
		where `AdminApp` is the scripting object, `update` is the command, `app1` is the name of the application you want to update, `file` is the content type, `operation` is an option of the **update** command, `update` is the value of the `operation`option, `contents` is an option of the **update** command, */apps/app1/my.xml* is the value of the `contents` option, `contenturi` is an option of the **update** command, *app1.jar/my.xml* is the value of the `contenturi` option.

processEmbeddedConfig	Use this option to process the embedded configuration data that is included in the application. This option does not required a value. If the application Enterprise Archive (EAR) file contains embedded configuration data, this option is the default setting. If not, the default setting is the nonprocessEmbeddedConfig option.	
preCompileJSPs	Specifies to precompile the JavaServer Pages files. This option does not require a value. The default is nopreCompileJSPs.	
reloadEnabled	Specifies that the file system of the application will be scanned for updated files so that changes reload dynamically. This option is the default setting and it does not require a value.	
reloadInterval	Specifies the time period in seconds that the file system of the application will be scanned for updated files. Valid range is greater than zero. The default is three seconds.	

server	Specifies the server name to install or update an entire application or to update an application in order to add a new module. If you want to update an application, this option only applies if the application contains a new module that does not exist in the installed application.	

update	Updates the installed application with a new version of the enterprise archive file (EAR) file. This option does not require a value. The application that is being updated, which is specified by the appname option, must already be installed in the WebSphere Application Server configuration. The update action merges bindings from the new version with the bindings from the old version, uninstalls the old version, and installs the new version. The binding information from new version of the EAR file is preferred over the corresponding one from the old version. If any element of binding is missing in the new version, the corresponding element from the old version is used.	

update.ignore.new	Specifies that during the update action, bindings from the new version of the application are ignored. This option does not require a value. This option applies only if you specify one of the following items: • The update option for the **install** command. • The modulefile or app as the content type for the **update** command.	
update.ignore.old	Specifies that during the update action, the bindings from the installed version of the application are ignored. This option does not require a value. This option applies only if you specify one of the following items: • The update option for the **install** command. • The modulefile or app as the content type for the **update** command.	

useMetaDataFromBinary	Specifies that the metadata that is used at run time, for example, deployment descriptors, bindings, extensions, and so on, come from the EAR file. This option does not require a value. The default value is `nouseMetaDataFromBinary`, which means that the metadata that is used at run time comes from the configuration repository.	
usedefaultbindings	Specifies to use default bindings for installation. This option does not require a value. The default setting is nousedefaultbindings.	

validateinstall	Specifies the level of application installation validation. Valid option values include: • off - Specifies no application deployment validation. This value is the default. • warn - Performs application deployment validation and continues with the application deployment process even when reported warnings or error messages exist. • fail - Performs application deployment validation and does not to continue with the application deployment process when reported warnings or error messages exist.	
verbose	Causes additional messages to display during installation. This option does not require a value.	

WebServicesClientBindDeployedWSDL	The immutable values for this option identify the client Web service that you are modifying. The scoping fields include: Module, EJB, and Web service. The single mutable value for this task is the deployed WSDL file name. It indicates the Web Services Description Language (WSDL) the client uses.
	The Module field identifies the enterprise or Web application within the application. If the module is an enterprise bean , the EJB field identifies a particular enterprise bean within the module. The Web service field identifies the Web service within the enterprise bean or the Web application module. This identifier corresponds to the wsdl:service attribute in the WSDL file, prepended with `service/`, for example, `service/WSLoggerService2`.
		Using Jacl:
		`$AdminApp install WebServicesSamples.ear`
		` {-WebServicesClientBindDeployedWSDL`
		`{{AddressBookW2JE.jar AddressBookW2JE service/WSLoggerService2`
		`META-INF/wsdl/DeployedWsdl1.wsdl}}}`
		Using Jython:
		`AdminApp.install('WebServicesSamples.ear',`
		` '[-WebServicesClientBindDeployedWSDL`
		`[[AddressBookW2JE.jar AddressBookW2JE service/WSLoggerService2`
		`META-INF/wsdl/DeployedWsdl1.wsdl]]]')`

WebServicesClientBind DeployedWSDL continued	The deployed WSDL attribute names a WSDL file relative to the client module. An example of a deployed WSDL for a Web application is the following: `WEB-INF/wsdl/WSLoggerService`.	

WebServicesClientBind PortInfo	The immutable values identify the port of a client Web service that you are modifying. The scoping fields include: Module, EJB, Web service and Port. The mutable values for this task include: `Sync Timeout`, `BasicAuth ID`, `BasicAuth Password`, `SSL Config`, and `Overridden Endpoint URI`. The basic authentication and Secure Sockets Layer (SSL) fields affect transport level security, not Web services security.	Using Jacl:
		`$AdminApp install WebServicesSamples.ear`
		` {-WebServicesClientBindPortInfo`
		`{{AddressBookW2JE.jar AddressBookW2JE`
		`service/WSLoggerService2 WSLoggerJMS 3000 newHTTP_ID newHTTP_pwd`
		`sslAliasConfig http://yunus:9090/WSLoggerEJB/services/WSLoggerJMS}}}`
		Using Jython:
		`AdminApp.install('WebServicesSamples.ear',`
		` '[-WebServicesClientBindPortInfo`
		`[[AddressBookW2JE.jar AddressBookW2JE`
		`service/WSLoggerService2 WSLoggerJMS 3000 newHTTP_ID newHTTP_pwd`
		`sslAliasConfig http://yunus:9090/WSLoggerEJB/services/WSLoggerJMS]]]')`

WebServicesClientBindPreferredPort	Associates a preferred port (implementation) with a port type (interface) for a client Web service. The immutable values identify a port type of the client Web service that you are modifying. The scoping fields include: Module, EJB, Web service and Port Type. The mutable value for this task is `Port`.
	• Port Type - QName ("{namespace} localname") of a port type that is defined by a wsdl:portType attribute in the WSDL file that identifies an interface.
	• Port - QName of a port defined by a wsdl:port attribute within a wsdl:service attribute in a WSDL file that identifies an implementation that has preference.
		Using Jacl:
		`$AdminApp install WebServicesSamples.ear {-WebServicesClientBindPreferredPort {{AddressBookW2JE.jar AddressBookW2JE service/WSLoggerService2 WSLoggerJMS WSLoggerJMSPort}}}`
		Using Jython:
		`AdminApp.install('WebServicesSamples.ear', '[-WebServicesClientBindPreferredPort [[AddressBookW2JE.jar AddressBookW2JE service/WSLoggerService2 WSLoggerJMS WSLoggerJMSPort]]]')`

WebServicesServerBindPort	Sets two attributes of a Web service port. The immutable values identify the port of a Web service that you are modifying. The scope fields include: Module, Web service and Port. The mutable values include: `WSDL Service Name`, and `Scope`.
	The scope determines the life cycle of implementing the Java bean. The valid values include: `Request` (new instance for each request), `Application` (one instance for each web-app), and `Session` (new instance for each HTTP session).
	The scope attribute does not apply to Web services that a Java Message Service (JMS) transport. The scope attribute does not apply to enterprise beans.
	The WSDL service name identifies a service when more than one service has the same port name. The WSDL service name is represented as a QName string, for example, `{namespace}localname` .
		Using Jacl:
		`$AdminApp install WebServicesSamples.ear {-WebServicesServerBindPort {{AddressBookW2JE.jar service/WSLoggerService2 WSLoggerJMS {} Session}}}`
		Using Jython:
		`AdminApp.install('WebServicesSamples.ear', '[-WebServicesServerBindPort [[AddressBookW2JE.jar service/WSLoggerService2 WSLoggerJMS "" Session]]]')`

WebServicesClient CustomProperty	Supports the configuration of the name value parameter for the description of the client bind file of a Web service. The immutable values identify the port of the Web service that you are modifying. The scope fields include: Module, Web service, and Port. The mutable values include: `name` and `value`.
	The format of the `name` and `value` values include a string that represents multiple name and value pairs by using the + character as a separator. For example, name string = ″n1+n2+n3″ value string = ″v1+v2+v3″ yields name/value pairs: {{″n1″ ″v1″}, {″n2″ ″v2″}, {″n3″ ″ v3″}}
		Using Jacl:
		`$AdminApp edit WebServicesSamples {-WebServicesServerCustomProperty`
		`{{AddressBookW2JE.jar AddressBookService AddressBook http.proxyHost`
		`+http.proxyPort myhost+80}}}`
		Using Jython:
		`AdminApp.edit ( 'WebServicesSamples', '[ -WebServicesServerCustomProperty`
		`[[AddressBookW2JE.jar AddressBookService AddressBook`
		`http.proxyHost+http.proxyPort myhost+80]]]')`

WebServicesServer CustomProperty	Supports the configuration of the name value parameter for the description of the server bind file of a Web service. The scoping fields include the following: Module, EJB, and Web service. The mutable values for this task include: `name` and `value`.
	The format of the these values include a string that represents multiple name and value pairs by using the plus (+) character as a separator. For example, name string = ″n1+n2+n3″ value string = ″v1+v2+v3″ yields name/value pairs: {{″n1″ ″v1″}, {″n2″ ″v2″}, {″n3″ ″ v3″}}
		Using Jacl:
		`$AdminApp edit WebServicesSamples {-WebServicesServerCustomProperty {{AddressBookW2JE.jar AddressBookService AddressBook com.ibm.websphere.webservices.http.responseContentEncoding deflate}}}`
		Using Jython:
		`AdminApp.edit ( 'WebServicesSamples', '[ -WebServicesServerCustomProperty [[AddressBookW2JE.jar AddressBookService AddressBook com.ibm.websphere.webservices.http.responseContentEncoding deflate]]]')`

Usage table for the options of the AdminApp object install, installInteractive, update, updateInteractive, edit, and editInteractive commands:

The following table lists all of the options available for the **install**, **installInteractive**, **update**, **updateInteractive**, **edit**, and **editInteractive** commands of the AdminApp object. The table indicates the applicable commands for each option.

Option name	install and install Interactive commands - install an application	update and update Interactive commands - Update an application	update and update Interactive commands - Add a module	update and update Interactive commands - Update a module	edit and edit Interactive commands - Edit an application	edit and edit Interactive commands - Edit a module
ActSpecJNDI	Yes	Yes	Yes	Yes	Yes	Yes
allowPermInFilterPolicy	Yes	Yes				
appname	Yes	Yes				
BackendIdSelection	Yes	Yes	Yes	Yes		

BindJndiForEJBMessagingBinding	Yes	Yes	Yes	Yes	Yes	Yes
BindJndiForEJBNonMessageBinding	Yes	Yes	Yes	Yes	Yes	Yes
cell	Yes	Yes	Yes			
cluster	Yes	Yes	Yes			
contents		Yes	Yes	Yes		
contenturi		Yes	Yes	Yes		
contextroot	Yes	Yes	Yes			
CorrectOracleIsolationLevel	Yes	Yes	Yes	Yes	Yes	Yes
CorrectUseSystemIdentity	Yes	Yes	Yes	Yes	Yes	Yes
createMBeansForResources	Yes	Yes			Yes	
custom	Yes	Yes	Yes	Yes	Yes	Yes
DataSourceFor10CMPBeans	Yes	Yes	Yes	Yes	Yes	Yes
DataSourceFor20CMPBeans	Yes	Yes	Yes	Yes	Yes	Yes
DataSourceFor10EJBModules	Yes	Yes	Yes	Yes	Yes	Yes
DataSourceFor20EJBModules	Yes	Yes	Yes	Yes	Yes	Yes
defaultbinding.datasource.jndi	Yes	Yes	Yes	Yes		
defaultbinding.cf.jndi	Yes	Yes	Yes	Yes		
defaultbinding.cf.resauth	Yes	Yes	Yes	Yes		
defaultbinding.datasource.password	Yes	Yes	Yes	Yes		
defaultbinding.datasource.username	Yes	Yes	Yes	Yes		
defaultbinding.ejbjndi.prefix	Yes	Yes	Yes	Yes		
defaultbinding.force	Yes	Yes	Yes	Yes		
defaultbinding.strategy.file	Yes	Yes	Yes	Yes		
defaultbinding.virtual.host	Yes	Yes	Yes	Yes		
depl.extension.reg (deprecated)						
deployejb	Yes	Yes	Yes	Yes		
deployejb.classpath	Yes	Yes	Yes	Yes		
deployejb.dbschema	Yes	Yes	Yes	Yes		
deployejb.dbtype	Yes	Yes	Yes	Yes		
deployejb.rmic	Yes	Yes	Yes	Yes		
deployws	Yes	Yes	Yes	Yes		
deployws.classpath	Yes	Yes	Yes	Yes		
deployws.jardirs	Yes	Yes	Yes	Yes		
distributeApp	Yes	Yes			Yes	
EmbeddedRar	Yes	Yes	Yes	Yes	Yes	Yes
EnsureMethodProtectionFor10EJB	Yes	Yes	Yes	Yes		
EnsureMethodProtentionFor20EJB	Yes	Yes	Yes	Yes		
installdir (deprecated)						
installed.ear.destination	Yes	Yes			Yes	
MapMessageDestinationRefToEJB	Yes	Yes	Yes	Yes	Yes	Yes
MapModulesToServers	Yes	Yes	Yes	Yes	Yes	Yes
MapEJBRefToEJB	Yes	Yes	Yes	Yes	Yes	Yes

MapResEnvRefToRes	Yes	Yes	Yes	Yes	Yes	Yes
MapResRefToEJB	Yes	Yes	Yes	Yes	Yes	Yes
MapRolesToUsers	Yes	Yes			Yes	Yes
MapRunAsRolesToUsers	Yes	Yes	Yes	Yes	Yes	Yes
MapWebModToVH	Yes	Yes	Yes	Yes	Yes	Yes
noallowPermInFilterPolicy	Yes	Yes				
nocreateMBeansForResources	Yes	Yes			Yes	
node	Yes	Yes	Yes			
nodeployejb	Yes	Yes	Yes	Yes		
nodeployws	Yes	Yes	Yes	Yes		
nodistributeApp	Yes	Yes			Yes	
nopreCompileJSPs	Yes	Yes	Yes	Yes		
noprocessEmbeddedConfig	Yes	Yes				
noreloadEnabled	Yes	Yes			Yes	
nousedefaultbindings	Yes	Yes	Yes	Yes		
nouseMetaDataFromBinary	Yes	Yes			Yes	
operation		Yes	Yes	Yes		
preCompileJSPs	Yes	Yes	Yes	Yes		
processEmbeddedConfig	Yes	Yes				
reloadEnabled	Yes	Yes			Yes	
reloadInterval	Yes	Yes			Yes	
server	Yes	Yes	Yes			
update	Yes	Yes				
update.ignore.old	Yes	Yes		Yes		
update.ignore.new	Yes	Yes		Yes		
useMetaDataFromBinary	Yes	Yes			Yes	
usedefaultbindings	Yes	Yes	Yes	Yes		
validateinstall	Yes				Yes	
verbose	Yes	Yes	Yes	Yes	Yes	Yes
WebServicesClientBindingDeployedWSDLs	Yes	Yes	Yes	Yes	Yes	Yes
WebServicesClientBindPortInfo	Yes	Yes	Yes	Yes	Yes	Yes
WebServicesClientBindPreferredPort	Yes	Yes	Yes	Yes	Yes	Yes
WebServicesClientCustomProperty	Yes	Yes	Yes	Yes	Yes	Yes
WebServicesServerBindPort	Yes	Yes	Yes	Yes	Yes	Yes
WebServicesServerCustomProperty	Yes	Yes	Yes	Yes	Yes	Yes

Example: Obtaining option information for AdminApp object commands: Use the **taskInfo** command of the AdminApp object to obtain information about the data that is needed for your application. You need to provide data for rows or entries that are either missing information, or require an update.

- You can use the **options** command to see the requirements for an enterprise archive file (EAR) file if you construct installation command lines. The **taskInfo** command provides detailed information for each task option with a default binding applied to the result.
- The options for the AdminApp **install** command can be complex if you specify various types of binding information, for example, Java Naming and Directory Interface (JNDI) name, data sources for enterprise

bean modules, or virtual hosts for Web modules. An easy way to specify command-line installation options is to use a feature of the **installInteractive** command that generates the options for you. After you install the application interactively once and specify all the updates that you need, look for message WASX7278I in the wsadmin output log. The default output log for wsadmin is `wsadmin.traceout`. You can cut and paste the data in this message into a script, and modify it. For example:

```
WASX7278I: Generated command line: install c:/websphere/appserver/installableapps/jmsample.ear
{-BindJndiForEJBNonMessageBinding {{deplmtest.jar MailEJBObject deplmtest.jar,META-INF/ejb-jar.xml ejb/JMSampEJB1 }}
-MapResRefToEJB {{deplmtest.jar MailEJBObject deplmtest.jar,META-INF/ejb-jar.xml mail/MailSession9
javax.mail.Session mail/DefaultMailSessionX } {"JavaMail Sample WebApp"  mtcomps.war,WEB-INF/web.xml
mail/MailSession9 javax.mail.Session mail/DefaultMailSessionY }} -MapWebModToVH {{"JavaMail Sample WebApp"
mtcomps.war,WEB-INF/web.xml newhost }} -nopreCompileJSPs -novalidateApp -installed.ear.destination
c:/mylocation -distributeApp -nouseMetaDataFromBinary}
```

Commands for the AdminTask object

Use AdminTask object to run an administrative command. Administrative commands are discovered dynamically when you start the wsadmin tool. The administrative commands that are available for your use, and what you can do with them, depends on the edition of the WebSphere Application Server that you have.

You can start the scripting client without a server running by using the `-conntype NONE` option with the wsadmin tool. The AdminTask administrative commands are available in both connected and local modes. If a server is currently running, it is not recommended to run the AdminTask commands in local mode. This is because any configuration changes made in local mode will not be reflected in the running server configuration and vice versa. If you save a conflicting configuration, you could corrupt the configuration. In a deployment manager environment, configuration updates are available only if a scripting client is connected to a deployment manager. When connected to a node agent or a managed application server, you will not be able to update the configuration because the configuration for these server processes are copies of the master configuration which resides in the deployment manager. The copies are created on a node machine when a configuration synchronization occurs between the deployment manager and the node agent. Make configuration changes to the server processes by connecting a scripting client to a deployment manager. For this reason, to change a configuration, do not run a scripting client in local mode on a node machine. It is not a supported configuration.

The following commands are available for the AdminTask object:

Command name:	Group name:	Description:	Target object:	Parameters and return values:	Examples:

addNode Group Member	Node Group Commands group	The **addNode Group Member** command adds a member to a node group. Nodes may be members of more than one node group. The command will do validity checking to ensure the following:
		• Distributed and z/OS nodes are not combined in the same node group.
			The target object is the node group where the member will be created. This target object is required.	• Parameters:
				- nodeName The name of the node to be added to a node group. This parameter is required.
				• Returns: Node group member object ID
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask addNodeGroupMember WBINodeGroup { -nodeName WBINode}`
					• Using Jython:
					`AdminTask.addNodeGroupMember( 'WBINodeGroup', '[-nodeName WBINode]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask addNodeGroupMember { -interactive}`
					• Using Jython:
					`AdminTask.addNodeGroupMember ( '[-interactive]')`

add SIBWS Inbound Port	SIB WebServices group	The **add SIBWS Inbound Port** command adds the configuration for an inbound port to an inbound service. This command will fail if:
		• the port name is already in use by another inbound port for the inbound service or the end point listener that you specified.
		• the template port that you specified does not exist in the template WSDL of the inbound service.
			The object name of the inbound service to which the port will be added.	• Parameters:
				name
				The name of the port. (required)
				endpointListener
				The name of the associated end point listener. (required)
				node
				The node where the endpoint listener is located. You must specify the node parameter, the server parameter, or the cluster parameter. (conditional)
				server
				The server where the endpoint listener is located. You must specify the node parameter, the server parameter, or the cluster parameter. (conditional)
				cluster
				The cluster where the endpoint listener is located. You must specify the node parameter, the server parameter, or the cluster parameter. (conditional)
				templatePort
				The name of the port in the template WSDL to use as a basis for the binding of the port. (optional)
				• Returns: The object name of the inbound port object that was created.
					Batch mode example usage:
					• Using Jacl:
					`set inPort [`
					`$AdminTask addSIBWSInboundPort`
					`$inService {`
					`-name "`*`MyServiceSoap`*`"`
					`-endpointListener "`*`SOAPHTTP1`*`"`
					`-node "`*`MyNode`*`"`
					`-server "`*`server1`*`"}]`
					• Using Jython:
					`inPort =`
					`AdminTask.addSIBWSInboundPort(`
					`inService, '[`
					`-name `*`MyServiceSoap`*
					`-endpointListener `*`SOAPHTTP1`*
					`-node `*`MyNode`*
					`-server `*`server1`*`]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask addSIBWSInboundPort {`
					`-interactive}`
					• Using Jython:
					`AdminTask.addSIBWSInboundPort (`
					`'[-interactive]')`

add SIBWS Outbound Port	SIB WebServices group	The **add SIBWS Outbound Port** command adds the configuration for an outbound port to an outbound service.	The object name of the outbound service for which the port will be associated.	• Parameters:
				name
				The name of the port in the WSDL of the service provider. (required)
				node
				Node where the port destination will be localized. You must specify the node parameter, the server parameter, or the cluster parameter. (conditional)
				server
				The server where the port destination will be localized. You must specify the node parameter, the server parameter, or the cluster parameter. (conditional)
				cluster
				The cluster where the port destination will be localized. You must specify the node parameter, the server parameter, or the cluster parameter. (conditional)
				destination
				The name of the port destination. (optional)
				userId
				The user ID to use to retrieve the WSDL. (optional)
				password
				The password to use to retrieve the WSDL. (optional)
				• Returns: The object name of the outbound port object that you created.
					Batch mode example usage:
					• Using Jacl:
					`set outPort [`
					` $AdminTask`
					` addSIBWSOutboundPort`
					` $outService {`
					`  -name "MyServiceSoap"`
					`  -node "MyNode"`
					`  -server "server"}]`
					• Using Jython:
					`outPort =`
					` AdminTask`
					`.addSIBWSOutboundPort(`
					`  outService, '[`
					`  -name MyServiceSoap`
					`  -node MyNode`
					`  -server server]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask`
					` addSIBWSOutboundPort {`
					` -interactive}`
					• Using Jython:
					`AdminTask`
					`.addSIBWSOutboundPort (`
					` '[-interactive]')`

add SIBus Member	SIB Admin Commands group	Use this command to add a server or a cluster to a SIB bus.	None	- Parameters: **bus** name of bus to add member to (String, required) **node** to specify a server bus member, supply node and server name, but not cluster name (String, optional) **server** to specify a server bus member, supply node and server name, but not cluster name (String, optional) **cluster** to specify a cluster bus member, supply cluster name but not node and server name (String, optional) **createDefault Datasource** set this to true if a default data source should be created when the messaging engine is created. (Boolean, optional) **datasource JndiName** the JNDI name of the data source to be referenced from the datastore created when the member is added to the bus (String, optional) - Returns:	**Batch mode example usage:** - Using Jacl: $AdminTask addSIBusMember { -bus *busname* -node *nodename* -server *servername* -description *text*} - Using Jython: AdminTask.addSIBusMember('[-bus *busname* -node *nodename* -server *servername* -description "*text*"]') **Interactive mode example usage:** - Using Jacl: $AdminTask addSIBusMember { -interactive} - Using Jython: AdminTask.addSIBusMember ('[-interactive]')

addWSGWTargetService	WSGateway group	The **add WSGW Target Service** command adds a target to a gateway service. You must specify the target Service parameter or the target Destination parameter.	Object name of the Gateway Service object	• Parameters:
				name
				The administrative name of the target service. (Required)
				target Destination
				The name of the target destination. This can be within the same bus as the gateway destination or in a different bus. If the target destination is not within the same bus as the gateway destination, you must also specify the targetBus parameter. You must either specify the target Destination parameter or the target Service parameter. (Conditional)
				target Service
				The name of the target outbound service. You must either specify the target Destination parameter or the target Service parameter. (Conditional)
				targetBus
				The name of the WPM bus that contains the target. (Optional)
				• Returns: The object name of the target service object that you created.
					Batch mode example usage:
					• Using Jacl:
					`set gwTarget [`
					` $AdminTask`
					` addWSGWTargetService`
					` $gwService {`
					`  -name "`*AnotherTarget*`"`
					`  -targetService`
					` "`*AnotherService*`"}]`
					• Using Jython:
					`gwTarget=`
					` AdminTask.`
					`addWSGWTargetService(`
					`  gwService, '[`
					`  -name `*AnotherTarget*
					`  -targetService`
					` `*AnotherService*`]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask`
					`addWSGWTargetService {`
					` -interactive}`
					• Using Jython:
					`AdminTask`
					`.addWSGWTargetService`
					` ('[ -interactive]')`

compareNodeVersion	ManagedObjectMetadata group	The **compareNodeVersion** command compares the WebSphere Application Server version given a node that you specify and an input version.	None	• Parameters:
				- nodeName
				The name of the node associated with the metadata you want this command to return.
				- version
				A version number that you want to compare to the WebSphere Application Server version number.
				• Returns:
				– 0 if node version matches the input version
				– -1 if node version is smaller than the input version
				– 1 is node version is higher than the input version
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask compareNodeVersion {-nodeName` *`node1`* `-version` *`5`*`}`
					• Using Jython:
					`AdminTask.compareNodeVersion('[-nodeName` *`node1`* `-version` *`5`*`]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask compareNodeVersion {-interactive}`
					• Using Jython:
					`AdminTask.compareNodeVersion ('[-interactive]')`

configureTAM					**Interactive mode example usage:**
					• Using Jacl:
					`$AdminTask configureTAM {-interactive}`
					• Using Jython:
					`AdminTask.configureTAM ('[-interactive]')`

connect SIBWS Endpoint Listener	SIB Web Services group	The **connect SIBWS Endpoint Listener** command connects an end point listener to a bus.	Object name of the end point listener that you want to create.	• Parameters:
				bus
				The name of the bus to which the end point listener will be connected. (required)
				replyDestination
				The name of the reply destination for the connection. (optional)
				• Returns: The SIBWS bus connection property object.
					Batch mode example usage:
					• Using Jacl:
					`set busConn [ $AdminTask connectSIBWSEndpointListener $epl {-bus "MyBus"}]`
					• Using Jython:
					`busConn = AdminTask .connectSIBWSEndpointListener (epl, '[-bus MyBus]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask connectSIBWSEndpointListener { -interactive}`
					• Using Jython:
					`AdminTask .connectSIBWSEndpointListener ('[-interactive]')`

copy Resource Adapter	JCA management group	Use the **copy Resource Adapter** command to create a Java 2 Connector (J2C) resource adapter under the scope that you specify.	J2CResource Adapter _object _ID	• Parameters:
				- name Indicates the name of the new J2C resource adapter. This parameter is required.
				- scope Indicates the scope object ID. This parameter is required.
				- use Deep Copy If you set this parameter to `true`, all of the J2C connection factory, J2C activation specification, and J2C administrative objects will be copied to the new J2C resource adapter (deep copy). If you set this parameter to `false`, the objects are not created (shallow copy). The default is `false`.
				• Returns: J2C resource adapter object ID
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask copyResourceAdapter $ra [subst { -name newRA -scope $scope}]`
					• Using Jython:
					`AdminTask .copyResourceAdapter( ra, '[-name newRA -scope scope]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask copyResourceAdapter { -interactive}`
					• Using Jython:
					`AdminTask .copyResourceAdapter ('[-interactive]')`

create Application Server					**Interactive mode example usage:**
					• Using Jacl:
					`$AdminTask createApplicationServer {-interactive}`
					• Using Jython:
					`AdminTask .createApplicationServer ('[-interactive]')`

create Application Server Template					**Interactive mode example usage:**
					- Using Jacl:
					`$AdminTask createApplicationServerTemplate {-interactive}`
					- Using Jython:
					`AdminTask .createApplicationServerTemplate ('[-interactive]')`

createChain	Channel Framework Management group	The **createChain** command creates a new chain of transport channels based on a chain template.	The instance of the transport channel service under which the new chain is created. (ObjectName, required)	- Parameters:
				- template
				The chain template on which to base the new chain. (ObjectName, required)
				- name
				The name of the new chain. (String, required)
				- endPoint
				The name of the end point to be used by the instance of the TCP inbound channel in the new chain if the chain is an inbound chain. (ObjectName, optional)
				- Returns: The object name of the channel chain that was created.
					Batch mode example usage:
					- Using Jacl:
					`$AdminTask`
					` createChain (`
					` cells/rohitbuildCell01`
					`/nodes`
					`/rohitbuildCellManager01`
					`/servers`
					` /dmgr|server.xml#`
					`TransportChannelService_1) {`
					` -template` *WebContainer*
					`(templates`
					`/chains|webcontainer`
					`-chains.xml#Chain_1)`
					`-name` *trialChain1* `}`
					`$AdminTask createChain (`
					` cells/rohitbuildCell01`
					`/nodes/`
					` rohitbuildCellManager01`
					`/servers`
					` /dmgr|server.xml#`
					`TransportChannelService_1)`
					` {-template`
					 WebContainer
					`(templates/chains`
					`|webcontainer`
					` -chains.xml#Chain_1)`
					`-name` *trialChain1*
					` -endPoint (`
					` cells/rohitbuildCell01/`
					`nodes`
					`/rohitbuildCellManager01`
					` |serverindex.xml`
					`#EndPoint_3) }`
					- Using Jython:
					`AdminTask.createChain`
					`('cells/rohitbuildCell01`
					`/nodes/`
					`rohitbuildCellManager01`
					`/servers`
					` /dmgr|server.xml#`
					`TransportChannelService_1',`
					` '[-template "WebContainer`
					`(templates`
					`/chains|webcontainer`
					` -chains.xml#Chain_1)"`
					`-name trialChain]')`
					`AdminTask.createChain`
					`('cells`
					` /rohitbuildCell01/nodes`
					` /rohitbuildCellManager01/`
					`servers/dmgr|server.xml`
					` #TransportChannelService_1',`
					` '[-template`
					`"WebContainer(templates`
					` /chains|webcontainer-chains.`
					`xml#Chain_1)" -name`
					` trialChain`
					` -endPoint "(cells/`
					`rohitbuildCell01/nodes`
					` /rohitbuildCellManager01`
					`|serverindex`
					`.xml#EndPoint_3)"]')`

createChain continued					**Interactive mode example usage:**
					• Using Jacl:
					`$AdminTask createChain {-interactive}`
					• Using Jython:
					`AdminTask.createChain ( '[-interactive]')`

create Cluster	Cluster Config Commands	The **create Cluster** command creates a new server cluster. A server cluster consists of a group of application servers which are referred to as cluster members. Optionally, a replication domain can be created for the new cluster, and an existing server can be included as the first cluster member.	None	• **Parameters for step one:**
				-clusterConfig
				Specifies the configuration of the new server cluster. This command step is required. The following parameters can be specified for this step.
				clusterName
				The name of the new server cluster. This parameter is required.
				preferLocal
				Enables or disables node scoped routing optimization within this cluster. This parameter is optional. The value is true or false. It not specified, the default value is true.
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask createCluster { -clusterConfig {{cluster1 true}}}`
					`$AdminTask createCluster { -clusterConfig {{cluster1 true}} -replicationDomain {{true}}}`
					`$AdminTask createCluster { -clusterConfig {{ cluster1 true}} -convertServer {{ server1 node1 "" "" ""}}}`
					• Using Jython:
					`AdminTask.createCluster('[ -clusterConfig [[cluster1 true]]]')`
					`AdminTask.createCluster('[ -clusterConfig [[cluster1 true]] -replicationDomain [[true]]]')`
					`AdminTask.createCluster('[ -clusterConfig [[cluster1 true]] -convertServer [[ server1 node1 "" "" ""]]]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask createCluster {-interactive}`
					• Using Jython:
					`AdminTask.createCluster ('[ -interactive]')`

create Cluster continued				**Parameters for step two:** **-replication Domain** Specifies the configuration of a replication domain for this cluster. A replication domain is used to support HTTP session data replication. This command step is optional. The following parameters can be specified for this step: **createDomain** Creates a replication domain with a name set to the name of the new cluster. This parameter is optional. The value is true or false. It not specified, the default value is false.	

create Cluster continued				**Parameters for step three:** **-convertServer** Specifies information about an existing application server to convert to be the first member of the cluster. This command step is optional. The following parameters can be specified for this step: **serverNode** The name of the node with the server to be converted to the first cluster member. This parameter is required for the command step. You must also specify the serverName parameter. **serverName** The name of the application server to be converted to the first cluster member. This parameter is required for the command step. You must also specify the serverNode parameter. **memberWeight** The weight of the cluster member. The weight controls the amount of work directed to the application server. If the weight is greater than the weight assigned to other cluster members, the server will receive a larger share of the workload. The value is a number between 0 and 100. If none is specified, the default is 2.	

create Cluster continued				**nodeGroup** The name of the node group which this cluster member‘s node, and all future cluster members‘ nodes, must belong to. All cluster members must reside on nodes in the same node group. This parameter is optional. If specified, it must be one of the node groups which this member‘s node belongs to. If not specified, the default value will be the first node group listed for this member‘s node. **replicatorEntry** Specifies a replicator entry for the converted member will be created in the cluster‘s replication domain. A replicator entry is used to provide HTTP session data replication. This command parameter is optional. The value is true or false which indicates whether the replicator entry will be created. The default value is false. You can specify this parameter only if the createDomain parameter was set to true in the replication Domain command step. **Returns:** ObjectName of cluster created.	

createClusterMember	Cluster Config Commands	The createClusterMember command creates a member of a server cluster. A cluster member is an application server that belongs to a cluster. If this is the first member of the cluster, you must specify a template to use as the model for the cluster member. The template can be either a default server template, or an existing application server	cluster Object ID - The configuration object ID of the cluster which the new member will belong to. If this is not specified, then the cluster Name parameter must be specified. The object name can be obtained programmatically via Java using the WebSphere Config Service API, or via wsadmin scripting using the Admin Config command.	• **Parameters:**
				-clusterName
				The name of the cluster which the new member will belong to. If this parameter is not specified, then the cluster object ID must be specified in the command target.
				• **Parameters for step one:**
				-memberConfig
				Specifies the attributes of the new cluster member to be created in the cluster. This command step is required. The following parameters can be specified for this step:
				memberName
				The name of the server to be created for the new cluster member. This parameter is required.
				memberNode
				The name of the node where the new cluster member will be created. This parameter is required.
					Batch mode example usage:
					• Using Jacl:
					First member creation using template name:
					`$AdminTask createClusterMember {`
					`-clusterName cluster1`
					`-memberConfig`
					`{{node1 member1 "" ""`
					`true false}}`
					`-firstmember {{`
					`serverTemplateName`
					`"" "" "" ""}}}`
					First member creation using server and node for template:
					`$AdminTask createClusterMember {`
					`-clusterName cluster1`
					`-memberConfig`
					`{{node1 member1 "" ""`
					`true false}}`
					`-firstmember {{`
					`"" node1 server1 "" ""}}}`
					Second member creation:
					`$AdminTask`
					`createClusterMember {`
					`-clusterName cluster1`
					`-memberConfig`
					`{{node1 member2 "" ""`
					`true false}}}`
					• Using Jython:
					First member creation using template name:
					`AdminTask.createClusterMember`
					`('[`
					`-clusterName cluster1`
					`-memberConfig`
					`[[node1 member1 "" ""`
					`true false]]`
					`-firstMember [[`
					`serverTemplateName`
					`"" "" "" ""]]]')`
					First member creation using server and node for template:
					`AdminTask.createClusterMember`
					`('[`
					`-clusterName cluster1`
					`-memberConfig`
					`[[node1 member1 "" ""`
					`true false]]`
					`-firstMember [[`
					`"" node1 server1`
					`"" ""]]]')`
					Second member creation:
					`AdminTask.createClusterMember`
					`('[`
					`-clusterName cluster1`
					`-memberConfig`
					`[[node1 member2 "" ""`
					`true false]]]')`

create Cluster Member continued				• **Parameters for step one continued:**
				memberWeight
				The weight of the new cluster member. This controls the amount of work directed to the application server. If the weight is greater than the weight assigned to other cluster members, the server will receive a larger share of the workload.
				genUniquePorts
				Generates unique port numbers for each HTTP transport defined in the server. The new server will not have HTTP transports which conflict with any other servers defined on the same node. The value is true or false. The default value is true .
				replicatorEntry
				Specifies a replicator entry for the new cluster member will be created in the cluster‘s replication domain. A replicator entry is used to provide HTTP session data replication. This command parameter is optional. The value is true or false which indicates whether the entry is created. Default value is false. Specify this parameter only if a replication domain has been created for the
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask createClusterMember {-interactive}`
					• Using Jython:
					`AdminTask.createClusterMember ('[-interactive]')`

create Cluster Member continued				• **Parameters for step two:** **-firstMember** Specifies additional information necessary to create the first cluster member. This command step is required when creating the first member of the cluster, and is executable only when creating the first member of the cluster. The target of this command step is a Boolean value indicating whether or not to perform this step. The default value is true if any of the step parameters are specified; otherwise the default value is false. The following parameters can be specified for this step: **templateName** The name of an application server template to use when creating the new cluster member. If you specify a template, you cannot specify the template ServerNode and template ServerName parameters to use an existing application server as a template. You are required to specify either the templateName,or the template ServerNode and template ServerName parameters in this step.	

create Cluster Member continued				• **Parameters for step two continued:** **template erverNode** The name of the node with an existing application server to use as the template when creating the new cluster member. If you specify the template ServerNode parameter, you must also specify the template ServerName parameter, and you cannot specify the templateName parameter. You are required to specify either the templateName parameter, or the template ServerNode and template ServerName parameters, in this step. **template ServerName** The name of the existing application server used as the model when creating a new cluster member. If you specify the template ServerName parameter, you must also specify the template ServerNode parameter, and cannot specify the templateName parameter. You must specify either the templateName parameter, or the template ServerNode and template ServerName parameters, in this command	

create Cluster Member continued				• **Parameters for step two continued:** **nodeGroup** The name of the node group which this cluster member‘s node, and all future cluster members‘ nodes, must belong to. All cluster members must reside on nodes in the same node group. This parameter is optional. If specified, it must be one of the node groups which this member‘s node belongs to. If not specified, the default value will be the first node group listed for this member‘s node. **coreGroup** The name of the core group this cluster member, and all future cluster members, must belong to. All cluster members must belong to the same core group. This parameter is optional. If not specified, the default value is the default core group defined in the cell. • **Returns:** ObjectName of cluster member created.	

create Core Group	Core Group Management group	The **create Core Group** command creates a new core group. The core group that you create will contain no members.	None	- Parameters:
				- coreGroupName The name of the core group that you are creating. (String required)
				- Returns: None
					Batch mode example usage:
					- Using Jacl:
					`$AdminTask createCoreGroup { -coreGroupName MyCoreGroup}`
					- Using Jython:
					`AdminTask.createCoreGroup('[ -coreGroupName MyCoreGroup]')`
					Interactive mode example usage:
					- Using Jacl:
					`$AdminTask createCoreGroup { -interactive}`
					- Using Jython:
					`AdminTask.createCoreGroup ('[ -interactive]')`

create Core Group Access Point	Core Group Bridge Management group	The **create Core Group Access Point** command creates a default core group access point for the core group that you specify and adds it to the default access point group. If the default access point group does not exist, the command creates a default access point group.	Core group bridge settings object for the cell. (ObjectName, required).	- Parameters:
				- coreGroupName The name of the core group for which the core group access point will be created. (String required)
				- Returns: None
					Batch mode example usage:
					- Using Jacl:
					`$AdminTask createCoreGroupAccessPoint (cells/rohitbuildCell01|coregroupbridge.xml#CoreGroupBridgeSettings_1) "-coreGroupName DefaultCoreGroup"`
					- Using Jython:
					`AdminTask.createCoreGroupAccessPoint('cells/rohitbuildCell01|coregroupbridge.xml#CoreGroupBridgeSettings_1', '[-coreGroupName DefaultCoreGroup]')`
					Interactive mode example usage:
					- Using Jacl:
					`$AdminTask createCoreGroupAccessPoint {interactive}`
					- Using Jython:
					`AdminTask.createCoreGroupAccessPoint ('[-interactive]')`

create Default CGAP					**Interactive mode example usage:**
					- Using Jacl:
					`$AdminTask createDefaultCGAP {-interactive}`
					- Using Jython:
					`AdminTask.createDefaultCGAP ('[-interactive]')`

create Generic Server	Serve r Management group
	Step: Config ProcDef
		Use the **create Generic Server** command to create a new generic server in the configuration. A generic server is a server that the WebSphere Application Server manages but did not supply. The **create Generic Server** command provides an additional step, Config ProcDef, that you can use to configure the parameters that are specific to generic servers.	None	• Parameters:
				- name
				The name of the server that you want to create.
				- templateName
				Picks up a server template. This step provides a list of application server templates for the node and server type. The default value is the default templates for the server type. (String, optional)
				- genUniquePorts
				The port for the server.
				- templateLocation
				The location of the server template.
				- startCommand
				Indicates the path to the command that will run when this generic server is started. (String, optional)
				- startCommand Args
				Indicates the arguments to pass to the startCommand when the generic server is started. (String, optional)
				- executable TargetKind
				Specifies whether a Java class name (use `JAVA_CLASS`) or the name of an executable JAR file (use `EXECUTABLE_JAR`) will be used as the executable target for this process. This field should be left blank for binary executables. This parameter is only applicable for
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask createGenericServer jim667BaseNode {-name jgeneric -ConfigProcDef {{ "/usr/bin/myStartCommand" "arg1 arg2" "" "" "/tmp/workingDirectory" "/tmp/stopCommand" "argy argz"}}}`
					• Using Jython:
					`AdminTask .createGenericServer( jim667BaseNode, '[-name jgeneric -ConfigProcDef [[ /usr/bin/myStartCommand "arg1 arg2" "" "" /tmp/workingDirectory /tmp/StopCommand "argy argz"]]]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask createGenericServer { -interactive}`
					• Using Jython:
					`AdminTask.createGenericServer ('[-interactive]')`

create Generic Server continued				• Parameters: **- executable Target** Specifies the name of the executable target (a Java class containing a main() method or the name of an executable JAR), depending on the executable target type. This field should be left blank for binary executables. This parameter is only applicable for Java processes. (String, optional) **- working Directory** Specifies the working directory for the generic server. **- stopCommand** Indicates the path to the command that will run when this generic server is stopped. (String, optional) **- stopCommand Args** Indicates the arguments to pass to the stopCommand parameter when the generic server is stopped. (String, optional) • Returns: null	

create Generic Server Template					**Interactive mode example usage:**
					• Using Jacl:
					`$AdminTask createGenericServerTemplate {-interactive}`
					• Using Jython:
					`AdminTask .createGenericServerTemplate ('[-interactive]')`

create J2C Activation Spec	JCA management group	Use the **create J2C Activation Spec** command to create a Java 2 Connector (J2C) activation specification under a J2C resource adapter and attributes that you specify. Use the message ListenerType parameter to indicate the activation specification that is defined for the J2C resource adapter.	J2C Resource Adapter _object _ID	• Parameters:
				- message ListenerType Identifies the activation specification for the J2C activation specification to be created. Use this parameter to identify the activation specification template for the J2C resource adapter that you specify.
				- name Indicates the name of the J2C activation specification that you are creating.
				- jndiName Indicates the name of the Java Naming and Directory Interface (JNDI).
				- destination JndiName Indicates the name of the Java Naming and Directory Interface (JNDI) of corresponding destination.
				- authentication Alias Indicates the authentication alias of the J2C activation specification that you are creating.
				- description Description of the created J2C activation spec.
				• Returns: J2CActivationSpec object ID
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask createJ2CActivationSpec $ra {-name J2CActSpec -jndiName eis/ActSpec1 -messageListenerType javax.jms.MessageListener }`
					• Using Jython:
					`AdminTask .createJ2CActivationSpec( ra, '[-name J2CActSpec -jndiName eis/ActSpec1 -messageListenerType javax.jms.MessageListener]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask createJ2CActivationSpec {-interactive}`
					• Using Jython:
					`AdminTask .createJ2CActivationSpec ('[-interactive]')`

create J2C Admin Object	JCA management group	Use the **create J2C Admin Object** command to create a administrative object under a resource adapter with attributes that you specify. Use the administrative object interface to indicate the administrative object defined in the resource adapter.	J2CResourceAdapter_object_ID	• Parameters:
				-adminObjectInterface
				Specifies the administrative object interface to identify the administrative object for the resource adapter that you specify. This parameter is required.
				-name
				Indicates the name of the administrative object.
				-jndiName
				Specifies the name of the Java Naming and Directory Interface (JNDI).
				-description
				Description of the created J2C admin object.
				• Returns: J2CAdminObject object ID
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask`
					` createJ2CAdminObject`
					` $ra {-adminObjectInterface`
					` fvt.adapter`
					`.message.FVTMessage`
					`Provider -name J2CA01`
					` -jndiName eis/J2CA01}`
					• Using Jython:
					`AdminTask`
					`.createJ2CAdminObject(`
					`ra, '[-adminObjectInterface`
					` fvt.adapter.message`
					`.FVTMessageProvider`
					` -name J2CA01 -jndiName`
					` eis/J2CA01]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask`
					` createJ2CAdminObject`
					` {-interactive}`
					• Using Jython:
					`AdminTask`
					`.createJ2CAdminObject`
					` ('[-interactive]')`

create J2C Connection Factory	JCA management group	Use the **create J2C Connection Factory** command to create a Java 2 connection factory under a Java 2 resource adapter and attributes that you specify. Use the connection factory interfaces to indicate the connection definitions defined for the Java 2 resource adapter.	J2C Connection Factory _object _ID	• Parameters:
				-connection Factory Interface
				Identifies the connection definition for the Java 2 resource adapter that you specify. This parameter is required.
				-name
				Indicates the name of the connection factory.
				-jndiName
				Indicates the name of the Java Naming and Directory Interface (JNDI).
				-description
				Description of the created J2C connection factory.
				• Returns: J2C connectionfactory object ID.
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask createJ2CConnectionFactory $ra {-connectionFactoryInterfaces javax.sql.DataSource -name J2CCF1 -jndiName eis/J2CCF1}`
					• Using Jython:
					`AdminTask. createJ2CConnectionFactory (ra, '[-connectionFactoryInterfaces javax.sql.DataSource -name J2CCF1 -jndiName eis/J2CCF1]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask createJ2CConnectionFactory {-interactive}`
					• Using Jython:
					`AdminTask .createJ2CConnectionFactory ('[-interactive]')`

create Node Group	Node Group Commands group	The **create Node Group** command creates a new node group. A node group consists of a group of nodes which are referred to as node group members. Optionally, you can create a short name and description for the new node group.	The node group name to be created. This target object is required.	• Parameters:
				- shortName
				The short name of the node group. This parameter is optional.
				- description
				The description of the node group. This parameter is optional.
				• Returns: Node group object ID
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask createNodeGroup WBINodeGroup`
					• Using Jython:
					`AdminTask .createNodeGroup( 'WBINodeGroup')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask createNodeGroup {-interactive}`
					• Using Jython:
					`AdminTask.createNodeGroup ('[-interactive]')`

create Node Group Property	Node Group Commands group	The **create Node Group Property** command creates custom properties for a node group.	The name of the node group. This target object is required.	• Parameters:
				- name
				The name of the custom property to create. This parameter is required.
				- value
				The value of the custom property. This parameter is optional.
				- description
				The description of the custom property. This parameter is optional.
				• Returns: Properties object ID
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask createNodeGroupProperty WBINodeGroup { -name Channel -value "channel1"}`
					• Using Jython:
					`AdminTask .createNodeGroupProperty( 'WBINodeGroup', '[-name Channel -value channel1]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask createNodeGroupProperty {-interactive}`
					• Using Jython:
					`AdminTask .createNodeGroupProperty ('[-interactive]')`

create SIB Destination	SIB Admin Commands	Use this command to create a SIB destination.	None	• Parameters: **bus** name of the bus where this destination is to be configured (String, required) **name** destination name (String, required) **type** The destination type. Valid value include: Queue, TopicSpace, WebService or Port. If the type is not TopicSpace, you must use the node/server or cluster option to specify a bus member. (String, required) **cluster** to assign the destination to a cluster, supply cluster name, but not node and server name. (optional) **node** to assign the destination to a server, supply node name server name, but not cluster name. (optional) **server** to assign the destination to a server, supply node name server name, but not cluster name. (optional) **aliasBus** if this is an alias destination, the source bus name of alias mapping. (optional)	**Batch mode example usage:** • Using Jacl: $AdminTask createSIBDestination {-bus *busname* -name *destname* -type TopicSpace} • Using Jython: AdminTask .createSIBDestination ('[-bus *busname* -name *destname* -type TopicSpace]') **Interactive mode example usage:** • Using Jacl: $AdminTask createSIBDestination {-interactive} • Using Jython: AdminTask .createSIBDestination ('[-interactive]')

create SIB Destination continued				• Parameters continued: **targetBus** if this is an alias destination, the name of the bus that the destination it maps to is configured on. (optional) **targetName** if this is an alias destination, the name of the destination it maps to. (optional) **foreignBus** if this is a foreign destination, the name of the foreign bus. (optional) **description** description. (optional) **reliability** the reliability quality of service for message flows through this destination, from BEST_EFFORT _NON-PERSISTENT to ASSURED _PERSISTENT, in order of increasing reliability. Higher levels of reliability have higher impacts on the performance. (optional) **maxReliability** the maximum reliability quality of service that is accepted for values specified by producers. (optional)	

create SIB Destination continued				**Parameters continued:** **override OfQOS ByProducer Allowed** controls the quality of service for message flows between producers and the destination. Select this option to use the quality of service specified by producers instead of the quality defined for the destination. (optional) **defaultPriority** the default priority for message flows through this destination, in the range 0 (lowest) through 9 (highest). This default priority is used for messages that do not contain a priority value (Integer, optional). (optional) **maxFailed Deliveries** the maximum number of times that service tries to deliver a message to the destination before forwarding it to the exception destination (Integer, optional). (optional) **exception Destination** the name of another destination to which the system sends a message that cannot be delivered to this destination within the specified maximum number of failed deliveries. (optional) **sendAllowed** clear this option (setting it to false) to stop producers from being able to send messages to this destination. (optional)	

create SIB Destination continued				**Parameters for step one continued:** **receiveAllowed** clear this option (setting it to false) to prevent consumers from being able to receive messages from this destination. (optional) **quiesceMode** select this option (setting it to true) to indicate that the destination is quiescing. In quiesce mode, new messages for the destination cannot be added to the bus, but any messages already in the bus can still be sent to, and processed by, the destination (Boolean, optional, default=False). (optional) **receiveExclusive** select this option (setting it to true) to allow only one consumer to attach to a destination (Boolean, optional, default=False). (optional) **topicAccess Check Required** topic access check required (Boolean, optional) **replyDestination** clear this option (setting it to false) to stop producers from being able to send messages to this destination. (optional) **replyDestination Bus** clear this option (setting to false) to prevent consumers from being able to receive messages from destination. (optional).	

create SIB Destination continued				**Parameters for step one:** **delegate Authorization CheckToTarget** indicates whether the authorization check should be delegated to the alias or target destination (Boolean, optional) **defaultForward RoutingPath** the default forward routing path. **bus** bus name **destination** destination name **Returns:** A new SIB destination.	

create SIB Engine	SIB Admin Commands	Use the **create SIB Engine** command to create a new messaging engine for a bus member.	None	• Parameters: **bus** name of the bus to which the messaging engine is to belong (String, optional) **node** to create a messaging engine on a server, supply node and server name, but not cluster name (String, optional) **server** to create a messaging engine on a server, supply node and server name, but not cluster name (String, optional) **cluster** to create a messaging engine on a cluster, supply cluster name, but not node and server name (String, optional) **description** description of the messaging engine (String, optional) **initialState** Indicates if the messaging engine is started or stopped when the associated application server starts. Until started, the messaging engine is unavailable. Valid values are `Stopped` and `Started`. (String, optional)	**Batch mode example usage:** • Using Jacl: $AdminTask createSIBEngine {-bus *busname* -node *nodeName* -server *severname*} • Using Jython: AdminTask.createSIBEngine ('[-bus *busname* -node *nodeName* -server *severname*]') **Interactive mode example usage:** • Using Jacl: $AdminTask createSIBEngine {-interactive} • Using Jython: AdminTask.createSIBEngine ('[-interactive]')

create SIB Engine continued				• Parameters continued: **destinationHighMsgs** the maximum total number of messages that the messaging engine can place on its message points (Long, optional) **createDefaultDatasource** Set to `true` if a default data source should be created when the messaging engine is created (Boolean, optional) **datasourceJndiName** JNDI name of the data source to be referenced from the datastore created when the messaging engine is created (String, optional) • Returns: A new SIB messaging engine.	

create SIB JMS Activation Spec	SIB JMS Admin Commands	Use the **create SIB JMS Activation Spec** command to create a SIB JMS activation specification.	Scope of the SIB JMS resource adapter to which the activation specification will be added.	• Parameters:
				name
				name of new activation specification (String, required)
				jndiName
				JNDI name of the activation specification (String, required)
				destinationJndi Name
				JNDI name of a destination (String, required)
				description
				a JMS activation specification is used by the default messaging provider to validate the activation-configuration properties for a JMS message-driven bean (MDB) (String, optional)
				acknowledge Mode
				how the session acknowledges any messages it receives (String, optional)
				authentication Alias
				authentication alias (String, optional)
				busName
				name of the SIB bus to connect to (String, required)
				clientId
				client identifier. Required for durable topic subscriptions (String, optional)
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask createSIBJMSActivationSpec $ra {-name` *`specname`* `-jndiName` *`specname`*`}`
					• Using Jython:
					`AdminTask .createSIBJMSActivationSpec (ra, '[-name` *`specname`* `-jndiName` *`specname`*`]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask createSIBJMSActivationSpec {-interactive}`
					• Using Jython:
					`AdminTask .createSIBJMSActivationSpec ('[-interactive]')`

create SIB JMS Activation Spec continued				• Parameters: **destinationType** Indicates if the message-driven bean uses a queue or topic destination. (String, optional) **durable Subscription Home** The name of the durable subscription home. This identifies the messaging engine where all durable subscriptions accessed through this activation specification are managed (String, optional) **maxBatchSize** the maximum number of messages received from the messaging engine in a single batch (Integer, optional) **maxConcurrency** the maximum number of endpoints to which messages are delivered concurrently (Integer, optional) **messageSelector** the JMS message selector used to determine which messages the message-driven bean (MDB) receives (String, optional) **password** password (String, optional)	

create SIB JMS Activation Spec continued				• Parameters: **subscription Durability** whether a JMS topic subscription is durable or nondurable (String, optional) **subscriptionName** the subscription name needed for durable topic subscriptions (String, optional) **shareDurable Subscriptions** used to control how durable subscriptions are shared (String, optional, default = AsCluster) **userName** user name (String, optional) • Returns: A new SIB JMS activation specification.	

create SIB JMS Connection Factory	SIB JMS Admin Commands	Use the **create SIB JMS Connection Factory** command to create a generic, queue or topic SIB JMS connection factory.	Scope of the SIB JMS resource adapter to which the SIB JMS connection factory will be added.	• Parameters:
				name
				The name of the SIB JMS connection factory (String, required)
				jndiName
				the JNDI name of the SIB JMS connection factory (String, required)
				type
				The type of connection factory to create. To create a queue connection factory, set the value to `Queue`. To create a topic connection factory, set to `Topic`. To create a generic connection factory, do not set a value. (String, optional)
				authDataAlias
				Specifies a user ID and password to be used to authenticate connections to the JMS provider for application-managed authentication (String, optional)
				category
				classifies or groups the connection factory (String, optional)
				description
				description of the connection factory (String, optional)
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask`
					`createSIBJMSConnectionFactory`
					` $ra {`
					`-name`
					`connectionfactory_name`
					` -jndiName` *`jndi_name`*`}`
					• Using Jython:
					`AdminTask`
					`.createSIBJMSConnectionFactory`
					`(ra, '[`
					`-name`
					`connectionfactory_name`
					` -jndiName`
					`jndi_name``]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask`
					`createSIBJMSConnectionFactory`
					` {-interactive}`
					• Using Jython:
					`AdminTask`
					`.createSIBJMSConnectionFactory`
					` ('[-interactive]')`

create SIB JMS Connection Factory continued				• Parameters: **logMissingTransactionContext** whether or not the container logs that there is a missing transaction context when a connection is obtained (Boolean, optional, default = False) **manageCachedHandles** Indicates if cached handles (handles held in instance variables in a bean) should be tracked by the container (Boolean, optional, default = False) **xaRecoveryAuthAlias** the authentication alias used during XA recovery processing (String, optional) **busName** the SIB bus name (String, optional) **clientID** user-defined string, only required for durable subscriptions (String, optional) **userName** The user name that is used to create connections from the connection factory (String, optional)	

create SIB JMS Connection Factory continued				- Parameters: **password** the password that is used to create connections from the connection factory (String, optional) **nonPersistent Mapping** non-persistent mapping value. Valid values include: `Best EffortNon Persistent`, `ExpressNon Persistent`, `ReliableNon Persistent`, `Reliable Persistent`, `Assured Persistent`, `AsSIB Destination` and `None` (String, optional) **persistent Mapping** persistent mapping value. Valid values include: `BestEffortNon Persistent`, `ExpressNon Persistent`, `ReliableNon Persistent`, `Reliable Persistent`, `Assured Persistent`, `AsSIBDestination` and `None` (String, optional) **durable Subscription Home** durable subscription home value (String, optional) **readAhead** read-ahead value. Valid values include: `Default`, `AlwaysOn` and `AlwaysOff` (String, optional)	

create SIB JMS Connection Factory continud				• Parameters: **target** the name of a target that resolves to a group of messaging engines (String, optional) **targetType** specifies the type of the name in the target parameter. Valid values are `BusMember`, `Custom` and `ME` (String, optional) **targetSignificance** this property specifies the significance of the target group (String, optional) **remoteProtocol** the name of the protocol that should be used to connect to a remote messaging engine (String, optional) **providerEnd Points** A list of endpoint triplets seperated by commas, for example: `host:port:chain` (String, optional) **connection Proximity** the proximity of acceptable messaging engines. Valid values include: `Bus`, `Host`, `Cluster` and `Server` (String, optional)	

create SIB JMS Connection Factory continud				• Parameters: **tempQueueName Prefix** temporary queue name prefix (String, optional) **tempTopicName Prefix** temporary topic name prefix (String, optional) **share DataSource WithCMP** used to control how data sources are shared (Boolean, optional) **shareDurable Subscriptions** used to control how durable subscriptions are shared. Legal values are ″AsCluster″, ″AlwaysShared″ and ″NeverShared″ (String, optional, default = AsCluster) • Returns: A new SIB JMS connection factory.	

create SIB JMS Queue	SIB JMS Admin Commands	Use the **create SIB JMS Queue** command to create a SIB JMS queue.	Scope of the SIB JMS resource adapter to which the SIB JMS queue will be added.	- Parameters:
				name
				The name of the SIB JMS queue. (String, required)
				jndiName
				The JNDI name of the SIB JMS queue. (String, required)
				description
				A description of the SIB JMS queue (String, optional)
				queueName
				The name of the underlying SIB queue to which the queue points (String, required)
				deliveryMode
				The delivery mode for messages. Legal values are ″Application″, ″NonPersistent″ and ″Persistent″ (String, optional)
				timeToLive
				the time in milliseconds to be used for message expiration (Long, optional)
				priority
				the priority for messages. Whole number in the range 0 to 9 (Integer, optional)
				readAhead
				read-ahead value. Legal values are ″AsConnection″, ″AlwaysOn″ and ″AlwaysOff″ (String, optional)
				busName
				the name of the bus on which the queue resides (String, optional)
				- Returns: A new SIB JMS queue.
					Batch mode example usage:
					- Using Jacl:
					`$AdminTask createSIBJMSQueue $ra {-name` *`queue_name`* `-jndiName` *`jndi_name`* `-queueName` *`queue_name`*`}`
					- Using Jython:
					`AdminTask .createSIBJMSQueue(ra, '[-name` *`queue_name`* `-jndiName` *`jndi_name`* `-queueName` *`queue_name`*`]')`
					Interactive mode example usage:
					- Using Jacl:
					`$AdminTask createSIBJMSQueue {-interactive}`
					- Using Jython:
					`AdminTask .createSIBJMSQueue ('[-interactive]')`

create SIB JMS Topic	SIB JMS Admin Commands	Use this command to create a SIB JMS topic.	Scope of the SIB JMS resource adapter to which the SIB JMS topic will be added.	• Parameters:
				name
				The name of the SIB JMS topic (String, required)
				jndiName
				the SIB JMS topic's JNDI name (String, required)
				description
				a description of the SIB JMS queue (String, optional)
				topicSpace
				the name of the underlying SIB topic space to which the topic points (String, required)
				***topicName**
				the topic to be used inside the topic space (for example, stock/IBM) (String, required)
				deliveryMode
				the delivery mode for messages. Legal values are ″Application″, ″NonPersistent″ and ″Persistent″ (String, optional)
				timeToLive
				the time in milliseconds to be used for message expiration (Long, optional)
				priority
				the priority for messages. Whole number in the range 0 to 9 (Integer, optional)
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask createSIBJMSTopic $ra {-name topic_name -jndiName jndi_name -topicName topic_name -topicSpace topicspace_name}`
					• Using Jython:
					`AdminTask .createSIBJMSTopic(ra, '[-name topic_name -jndiName jndi_name -topicName topic_name -topicSpace topicspace_name]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask createSIBJMSTopic {-interactive}`
					• Using Jython:
					`AdminTask .createSIBJMSTopic ('[-interactive]')`

create SIB JMS Topic continued				• Parameters continued: **readAhead** read-ahead value. Legal values are ″AsConnection″, ″AlwaysOn″ and ″AlwaysOff″ (String, optional) **busName** the name of the bus on which the topic resides (String, optional) • Returns: A new SIB JMS topic.	

create SIB Mediation	SIB Admin Commands	Use this command to create a SIB mediation.	None	• Parameters: **bus** name of the bus where the mediation is to be created (String, required) **mediationName** name to be given to the mediation (String, required) **description** description of the mediation (String, optional) **handler ListName** name of the handler list that was defined when the mediation was deployed (String, required) **globalTransaction** whether or not a global transaction is started for each message processed (Boolean, optional, default = False) **allow Concurrent Mediation** whether or not to apply the mediation to multiple messages concurrently, and preserve message ordering (Boolean, optional, default = False)	**Batch mode example usage:** • Using Jacl: $AdminTask createSIBMediation {-bus *bus_name* -mediationName *mediation_name* -handlerListName *handlerList_name*} • Using Jython: AdminTask .createSIBMediation('[-bus *bus_name* -mediationName *mediation_name* -handlerListName *handlerList_name*]') **Interactive mode example usage:** • Using Jacl: $AdminTask createSIBMediation {-interactive} • Using Jython: AdminTask .createSIBMediation ('[-interactive]')

create SIB Mediation continued				- Parameters: **discriminator** the text string that must not be present in a message for the mediation to process the message (String, optional) - Returns: A new SIB mediation.	

create SIBWS Endpoint Listener	SIB Web Services group	The **create SIBWS Endpoint Listener** command creates an end point listener object with no SIBWS bus connection property objects.	Object name of the server where the end point listener will be created.	- Parameters:
				name The name of the end point listener within the server. (required)
				urlRoot The root of the end point address URL for Web services that you access through the end point listener. (required)
				wsdlUrlRoot The root of the HTTP URL where you can retrieve the WSDL associated with this end point listener. (required)
				- Returns: The SIBWS end point object.
					Batch mode example usage:
					- Using Jacl:
					`set epl [$AdminTask createSIBWSEndpointListener $server {-name "soaphttp1" -urlRoot "http://myserver.com /wsgwsoaphttp1" -wsdlUrlRoot "http://myserver.com /wsgwsoaphttp1"}]`
					- Using Jython:
					`epl = AdminTask .createSIBWSEndpointListener (server, '[-name soaphttp1 -urlRoot http://myserver.com /wsgwsoaphttp1 -wsdlUrlRoot http://myserver.com /wsgwsoaphttp1]')`
					Interactive mode example usage:
					- Using Jacl:
					`$AdminTask createSIBWSEndpointListener {-interactive}`
					- Using Jython:
					`AdminTask .createSIBWSEndpointListener ('[-interactive]')`

create SIBWS Inbound Service	SIB Web Services group	The **create SIBWS Inbound Service** command creates a new inbound service object that represents a protocol attachment that service requesters will use. If you specify the UDDI Reference option, the wsdlLocation option is assumed to be a UDDI service key in the following format where each `n` is a hex digit: `nnnnnnnn nnnn-nnnn -nnnn-nnnn -nnnnnnnn`.	The object name of the messaging bus within which the service will be created.	• Parameters:
				name
				The administrative name of the inbound service. (required)
				destination
				The name of the underlying WPM destination. (required)
				wsdlLocation
				The location of the template WSDL. The value of this parameter can be a URL or a UDDI service key (UUID). (required)
				wsdl ServiceName
				The name of the service in the WSDL. You must specify this parameter or the wsdlServiceNamespace parameter. (conditional)
				wsdlService Namespace
				The namespace of the service in the WSDL. You must specify this parameter or the wsdlServiceName parameter. (conditional)
				uddiReference
				The reference of the UDDI registry for the WSDL. (optional)
				userId
				The user ID to use to retrieve the WSDL. (optional)
				password
				The password to use to retrieve the WSDL. (optional)
				• Returns: The object name of the created inbound service object.
					Batch mode example usage:
					• Using Jacl:
					`set inService [$AdminTask createSIBWSInboundService $bus {-name "`*`MyService`*`" -destination $destName -wsdlLocation "`*`http://myserver.com/MyService.wsdl`*`"}]`
					• Using Jython:
					`inService = AdminTask.createSIBWSInboundService(bus, '[-name `*`MyService`*` -destination `*`destName`*` -wsdlLocation `*`http://myserver.com/MyService.wsdl`*`]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask createSIBWSInboundService {-interactive}`
					• Using Jython:
					`AdminTask.createSIBWSInboundService('[-interactive]')`

create SIBWS Outbound Service	SIB Web Services group	The **create SIBWS Outbound Service** command creates a new outbound service object that represents a protocol attachment to a service provider. This command requires the identification of a single service element within a WSDL document.	The object name of the messaging bus within which the service is created.	- Parameters:
				name The administrative name of the outbound service. (required)
				wsdlLocation The location of the WSDL of the service provider. It can be a URL or a UDDI service key (UUID). (required)
				wsdlServiceName The name of the service in the WSDL. You must specify the wsdl ServiceName parameter or the wsdlService Namespsace parameter. (conditional)
				wsdlServiceNamespace Namespace of the service in the WSDL. You must specify the wsdlServiceName parameter or the wsdlService Namespsace parameter. (conditional)
				uddiReference The reference of the UDDI registry for the WSDL. (optional)
				destination The name of the service destination. (optional)
				userId The user ID to use to retrieve the WSDL. (optional)
				password The password to use to retrieve the WSDL. (optional)
				- Returns: The object name of the
					Batch mode example usage:
					- Using Jacl:
					`set outService [$AdminTask`
					` createSIBWSOutboundService`
					` $bus {-name`
					` "`*`MyService`*`"`
					` -wsdlLocation`
					` "`*`http://myserver.com`*
					`/MyService.wsdl``"}]`
					- Using Jython:
					`outService =`
					`  AdminTask`
					`.createSIBWSOutboundService`
					`(bus, '[-name `*`MyService`*
					` -wsdlLocation`
					` http://myserver.com`
					`/MyService.wsdl``]')`
					Interactive mode example usage:
					- Using Jacl:
					`$AdminTask`
					`createSIBWSOutboundService`
					` {-interactive}`
					- Using Jython:
					`AdminTask`
					`.createSIBWSOutboundService`
					` ('[-interactive]')`

create SI Bus	SIB Admin Commands	Use this command to create a new bus on the current node.	None	• Parameters:
				bus
				name of bus to create, which must be unique in the cell (String, required)
				description
				descriptive information about the bus (String, required)
				secure
				enable or disable bus security (Boolean, optional, default = False)
				interEngine AuthAlias
				name of the authentication alias used to authorize communication between messaging engines on the bus (String, optional)
				mediationsAuth Alias
				name of the authentication alias used to authorize mediations to access the bus (String, optional)
				protocol
				the protocol used to send and receive messages between messaging engines, and between API clients and messaging engines (String, optional)
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask createSIBus {-bus busname -description text -secure True -mediationsAuthAlias name -protocol protocol -discardOnDelete False}`
					• Using Jython:
					`AdminTask.createSIBus ('[-bus busname -description "text" -secure True -mediationsAuthAlias name -protocol protocol -discardOnDelete False]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask createSIBus {-interactive}`
					• Using Jython:
					`AdminTask.createSIBus ('[-interactive]')`

create SI Bus continued				- Parameters: **discardOnDelete** indicate whether or not any messages left in the data store of a queue should be discarded when the queue is deleted (Boolean, optional, default = False) **destinationHighMsgs** the maximum number of messages that any queue on the bus can hold (Long, optional) **configuration Reload Enabled** indicate whether configuration files should be dynamically reloaded for this bus (Boolean, optional, default = True) - Returns: A new SIB bus.	

createServerType	None	Use the **create Server Type** command to define a server type.	None	• Parameters:
				-version
				(String, required)
				-serverType
				(String, required)
				-createTemplateCommand
				(String, required)
				-createCommand
				(String, required)
				-defaultTemplateName
				The default value is: default. (String, optional)
				-configValidator
				(String, optional)
				• Returns: The identification of the server type that you created, `javax.management.ObjectName`.
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask createServerType {-version version -serverType serverType -createTemplateCommand name -createCommand name}`
					• Using Jython:
					`AdminTask.createServerType ('[-version version -serverType serverType -createTemplateCommand name -createCommand name]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask createServerType {-interactive}`
					• Using Jython:
					`AdminTask.createServerType ('[-interactive]')`

create TCP End Point	None	The **create TCP End Point** command creates a new named end point you can associate with a TCP inbound channel.	Parent instance of the Transport Channel Service that contains the TCP Inbound Channel. (Object Name, required)	- Parameters:
				-name
				Name for the new Named EndPoint. (String, required)
				- host
				Host for the new Named EndPoint. (String, required)
				- port
				Port for the new NamedEndPoint. (String, required)
				- Returns: Object name of the created NamedEndPoint.
					Batch mode example usage:
					- Using Jacl:
					`$AdminTask createTCPEndPoint (cells/rohitbuildCell01 /nodes/ rohitbuildCellManager01 /servers/dmgr |server.xml #TransportChannelService_1)`
					`{-name Sample_End_Pt_Name -host rohitbuild.raleigh.ibm.com -port 8978}`
					- Using Jython:
					`AdminTask.createTCPEndPoint ('cells/rohitbuildCell01 /nodes /rohitbuildCellManager01 /servers /dmgr|server.xml #TransportChannelService_1', '[-name Sample_End_Pt_Name -host rohitbuild.raleigh.ibm.com -port 8978]')`
					Interactive mode example usage:
					- Using Jacl:
					`$AdminTask createTCPEndPoint {-interactive}`
					- Using Jython:
					`AdminTask.createTCPEndPoint ('[-interactive]')`

create Unmanaged Node	Unmanaged Node Commands group	Use the **create Unmanaged Node** command to create a new unmanaged node in the configuration. An unmanaged node is a node that does not have a node agent nor a deployment manager. Unmanaged nodes may contain Web servers, such as IBM IHS server.	None	• Parameters:
				- nodeName
				The name that will represent the node in the configuration repository. (String, required)
				- hostName
				The host name of the system associated with this node. (String, required)
				- nodeOperating System
				The operating system in use on the system associated with this node. Valid entries include the following: `os400`, `aix`, `hpux`, `linux`, `solaris`, `windows`, and `os390`.(String required)
				• Returns: null
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask createUnmanagedNode {-nodeName myNode -hostName myHost -nodeOperatingSystem linux}`
					• Using Jython:
					`AdminTask .createUnmanagedNode ('[-nodeName jjNode -hostName jjHost -nodeOperatingSystem linux]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask createUnmanagedNode {-interactive}`
					• Using Jython:
					`AdminTask .createUnmanagedNode ('[-interactive]')`

create WSGW Gateway Service	WS Gateway group	The **create WSGW Gateway Service** command creates a new Gateway Service with associated Inbound Service and Target Service object. Configuration of the Inbound Port and Outbound Service/Port associated with these is done using separate commands.	Object Name of the gateway instance which the gateway service is created	- Parameters:
				-name Administrative name of the Gateway Service. (required)
				-wsdlLocation Location of the template WSDL. May be a URL or a UDDI business key (UUID). (conditional)
				-wsdl ServiceName The name of the service in the WSDL. (conditional)
				-wsdlService Namespace The namespace of the service in the WSDL. (conditional)
				-targetDestination The name of the target destination. (conditional)
				-targetService The name of the target outbound service. (conditional)
				-request Destination The name of the gateway destination. (optional)
				-replyDestination The name of the gateway reply destination. (optional)
				-targetBus The name of the WPM bus containing the target. (optional)
					Batch mode example usage:
					- Using Jacl:
					`set gwService [$AdminTask createWSGWGatewayService $wsgw {-name` *`MyGatewayService`* `-targetService` *`MyService`*`}]`
					- Using Jython:
					`gwService = AdminTask .createWSGWGatewayService (wsgw, '[-name` *`MyGatewayService`* `-targetService` *`MyService`*`]')`
					Interactive mode example usage:
					- Using Jacl:
					`$AdminTask createWSGWGatewayService {-interactive}`
					- Using Jython:
					`$AdminTask createWSGWGatewayService ('[-interactive]')`

create WSGW Gateway Service continued				• Parameters: **-uddiReference** The reference of the UDDI registry for the WSDL. (optional) **-userId** The user id to use to retrieve the WSDL. (optional) **-password** The password to use to retrieve the WSDL. (optional) • Returns: ObjectName of the created GatewayService object	

create WSGW Proxy Service	WS Gateway group	The **create WSGW Proxy Service** command creates a new proxy service with an associated inbound service, and a target service object with an associated outbound service. Configuration of the inbound port objects associated with the inbound service is done using separate commands.	The object name of the gateway instance within which the proxy service is created.	• Parameters:
				name The administrative name of the proxy service. (required)
				node The node where the destinations will be localized. (conditional)
				server The server where the destinations will be localized. (conditional)
				cluster Cluster where the destinations will be localized. (conditional)
				-requestDestination The name of the proxy request destination. (optional)
				-replyDestination The name of the proxy reply destination. (optional)
				-wsdlLocation The location of the proxy WSDL (URL). (optional)
				• Returns: The object name of the proxy service object that you created.
					Batch mode example usage:
					• Using Jacl:
					`set proxyService [$AdminTask createWSGWProxyService $wsgw {-name` *`MyProxyService`* `-node` *`MyNode`* `-server` *`server1`*`}]`
					• Using Jython:
					`proxyService = AdminTask .createWSGWProxyService (wsgw, '[-name` *`MyProxyService`* `-node` *`MyNode`* `-server` *`server1`*`]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask createWSGWProxyService {-interactive}`
					• Using Jython:
					`AdminTask .createWSGWProxyService ('[-interactive]')`

create Web Server	Server Management group	Use the **create Web Server** command to create a Web server definition. This command is a two step process. The first step creates a Web server definition using a template. The parameters of the second step configure the Web server definition properties. Web server definitions generate and propagate the `plugin-config.xml` file for each Web server. For IHS only, the Web server definitions allow you to administer and configure IHS Web servers using the administrative console. These functions include the following: Start, Stop, View logs, View and Edit configuration file.	None	- **Parameters for step one:**
				nodeName
				The name of the node. (String, required)
				name
				The name of the server. (String, required)
				templateName
				The name of the template that you want to use. Templates include the following: `IHS`, `iPlanet`, `IIS`, `DOMINO`, `APACHE`. The default template is `IHS`. (String, required)
				genUniquePorts
				Indicates that you want to generate unique ports. (optional)
				- **Parameters for step two:**
				serverConfig
				Create the Web server. (String, required)
				webPort
				The port for the Web server. (String, required)
				configurationFile
				The configuration file. The default is the path relative to the installation root, for example, `conf/httpd.conf`. (String, optional)
				webInstallRoot
				The installation path for the Web server. (String, required)
					Batch mode example usage:
					- Using Jacl:
					`$AdminTask createWebServer {-name web1 -serverConfig {{webPort WebserverInstallRoot PluginInstallRoot Configuration _file_name Windows_Server_Name errorLogPath accessLogPath WebProtocol}} -remoteServerConfig {{AdminPort UserID Password AdminProtocol}}`
					- Using Jython:
					`AdminTask.createWebServer ('[-name web1 -serverConfig [[webPort WebserverInstallRoot PluginInstallRoot Configuration _file_name Windows_Server _Name errorLogPath accessLogPath WebProtocol]] -remoteServerConfig [[AdminPort UserID Password AdminProtocol]]]')`
					where `-serverConfig` is second step of the command.
					– WebPort - is the port for the Webserver (required for all webservers)
					– Webserver InstallRoot - is the install path (directory) for webserver. necessary for IHS Admin Function.
					– Plugin Install Root - is install root where the plugin for the webserver is installed. Necessary for all webservers.
					– Configuration file name - is the file path for the IBM HTTP Server. This is necessary for View and edit of the IHS Configuration file only.

create Web Server continued				• **Parameters for step two continued:**
				pluginInstallRoot
				The plug-in installation path. (String, required)
				serviceName
				The service name. (String, optional)
				errorLogfile
				The error log for viewing. The default is the path relative to the installation root, for example, `logs/error_log`. (String, optional)
				accessLogfile
				The access log for viewing. The default is the path relative to the installation root, for example, `logs /access_log`. (String, optional)
				webProtocol
				Parameters for the IHS administration server running with an unmanaged or remote Web server. Options include `HTTP` or `HTTPS`. The default is `HTTP`. (String, required)
				adminPort
				The port of the IHS administrative server. (String, required)
				adminUserID
				The user ID. This value should match the one for authentication in the `admin.conf`. (String, required)
					• Windows Service Name - The windows service name on which IHS is to be started. This is necessary for Start and stop of the IHS webserver only.
					• ErrorLogPath - This is the path for the IHS error log (error.log)
					• AccessLogPath - This is the path for the IHS access log (access.log)
					• WebServerProtocol - HTTP or HTTPS
					where -remoteServerConfig is 3rd step of the command
					These parameters are only necessary if the IHS webserver is installed on a machine remote from WebSphere.
					• Admin Server Port - This is the port for the ADministration server. The administration server is installed on the same machine as the IBM HTTP Server. The admin server handles admin request to the IHS webserver.
					• UserID - This is the userID for authentication, if authentication is activated on the Administration server in the admin configuration file (admin.conf).
					• Passwd - This is the password for the specified authentication UserID. The password is generated by htpasswd utility in the admin.passwd file.
					• Admin ServerProtocol - HTTP or HTTPS
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask createWebServer -interactive`
					• Using Jython:
					`AdminTask.createWebServer ('[-interactive]')`

create Web Server continued				• **Parameters for step two continued:** **adminPasswrd** The administrative password. (String, required) **adminProtocol** The administrative protocol title. Options include `HTTP` or `HTTPS`. The default is `HTTP`. (String, required) • **Parameters for step three:** Parameters for IHS administration server running with an unmanaged or remote Web server (installed on machine different from WebSphere Application Server) **adminPortTitle (adminPort)** Port of IHS administration **admin UserIDTitle (adminUserID)** The user ID. This value should match the authentication in the admin.conf file. **adminPasswd Title (adminPasswd)** password **Admin Protocol Title (adminProtocol)** This parameter is required. The value is either HTTP or HTTPS. The default value is HTTP. • **Returns:** None	

delete Chain	Channel Framework Management group	The **delete Chain** command deletes an existing chain and, optionally, the transport channels in the chain.	The chain to be deleted. (Object Name ,required)	- Parameters:
				- delete Channels
				If the value of this attribute is true, non-shared transport channels used by the specified chain will be deleted. (Boolean, optional)
				- Returns: None
					Batch mode example usage:
					- Using Jacl:
					`$AdminTask deleteChain trialChain1 (cells/rohitbuildCell01 /nodes /rohitbuildCellManager01 /servers /dmgr|server.xml#Chain _1093554462922)`
					`$AdminTask deleteChain trialChain (cells/rohitbuildCell01 /nodes /rohitbuildCellManager01 /servers /dmgr|server.xml #Chain_1093554378078) {-deleteChannels true}`
					- Using Jython:
					`AdminTask.deleteChain ('trialChain1 (cells/rohitbuildCell01 /nodes /rohitbuildCellManager01 /servers/dmgr |server.xml #TransportChannelService_1)')`
					`AdminTask.deleteChain ('trialChain1 (cells/rohitbuildCell01 /nodes /rohitbuildCellManager01 /servers /dmgr |server.xml #TransportChannelService_1) ', '[-deleteChannels true]')`
					Interactive mode example usage:
					- Using Jacl:
					`$AdminTask deleteChain {-interactive}`
					- Using Jython:
					`AdminTask.deleteChain ('[-interactive]')`

delete Cluster	Cluster Config Commands	The delete Cluster command deletes the configuration of a server cluster. A server cluster consists of a group of application servers which are referred to as cluster members. When a server cluster is deleted, all of its members are deleted.
		Use the delete Cluster Member command to delete the configuration of an individual cluster member.
			cluster Object ID - The configuration object ID of the cluster to be deleted. If the cluster‘s object ID is not specified, then the cluster Name parameter must be specified. The object name can be obtained programmatically via Java using the WebSphere Config Service API, or via wsadmin scripting using the AdminConfig command.	• **Parameters:**
				-clusterName
				The name of the cluster to be deleted. If this parameter is not specified, then the cluster object ID must be specified in the command target.
				• **Parameters for step one:**
				-replication Domain
				Specifies the removal of the replication domain for this cluster. This command step is optional. The following parameters can be specified for this step:
				deleteDomain
				Deletes the replication domain for this cluster. This parameter is optional. The value is true or false which indicates whether the domain will be deleted. The default value is false. . Deleting the replication domain deletes all replicator entries defined in the domain.
				• **Returns:** None
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask deleteCluster { -clusterName cluster1 }`
					`$AdminTask deleteCluster { -clusterName cluster1 -replicationDomain {{true}}}`
					• Using Jython:
					`AdminTask.deleteCluster ('[-clusterName cluster1]')`
					`AdminTask.deleteCluster ('[-clusterName cluster1 -replicationDomain [[true]]]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask deleteCluster -interactive`
					• Using Jython:
					`AdminTask.deleteCluster ('[-interactive]')`

delete Cluster Member	Cluster Config Commands	The **delete Cluster Member** command deletes the configuration of a cluster member. A cluster member is an application server that belongs to a server cluster.
		Use the **delete Cluster** command to delete the configuration of a cluster.
			member Object ID - The configuration object ID of the cluster member to be deleted. If this is not specified, then the cluster Name, member Node and member Name parameters must be specified. The object name can be obtained programmatically via Java using the WebSphere Config Service API, or via wsadmin scripting using the Admin Config command.	• **Parameters:**
				-clusterName
				The name of the cluster which the member to be deleted belongs to. If this parameter is specified, then the member Name and member Node parameters must also be specified. If this is not specified, then the member object ID must be specified in the command target.
				-memberName
				The server name of the member to be deleted from the cluster. If this parameter is specified, then the clusterName and memberNode parameters must also be specified. If this is not specified, then the member object ID must be specified in the command target.
				-memberNode
				The name of the node having the cluster member to be deleted. If this parameter is specified, then the memberName and clusterName parameters must also be specified. If this is not specified, then the cluster member object ID must be specified in the command target.
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask deleteClusterMember {-clusterName cluster1 -memberNode node1 -memberName member1}`
					`$AdminTask deleteClusterMember {-clusterName cluster1 -memberNode node1 -memberName member2 -replicationEntry {{true}}}`
					• Using Jython:
					`AdminTask .deleteClusterMember ('[-clusterName cluster1 -memberNode node1 -memberName member1]')`
					`AdminTask .deleteClusterMember ('[-clusterName cluster1 -memberNode node1 -memberName member2 -replicationEntry [[true]]]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask deleteClusterMember -interactive`
					• Using Jython:
					`AdminTask .deleteClusterMember ('[-interactive]')`

delete Cluster Member continued				• **Parameters for step one:** **-replicatorEntry** Specifies the removal of a replicator entry for this cluster member. This command step is optional. The following parameters can be specified for this step: **deleteEntry** Delete the replicator entry having this cluster member‘s name from the cluster‘s replication domain. This parameter is optional. The value is true or false which indicates whether to delete the replicator entry. The default value is false. • **Returns:** None	

delete Core Group	Core Group Management group	The **delete Core Group** command deletes an existing core group. The core group that you specify must not contain any members. You cannot delete the default core group.	None	• Parameters:
				- core GroupName The name of the existing core group that will be deleted. (String required)
				• Returns: None
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask deleteCoreGroup {-coreGroupName *MyCoreGroup*}`
					• Using Jython:
					`AdminTask .deleteCoreGroup ('[-coreGroupName *MyCoreGroup*]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask deleteCoreGroup {-interactive}`
					• Using Jython:
					`AdminTask .deleteCoreGroup ('[-interactive]')`

delete Core Group Access Points	Core Group Bridge Management group	The **delete Core Group Access Points** command deletes all the core group access points associated with a group that you specify.	Core group bridge settings object for the cell. (Object Name, required)	• Parameters: **- core GroupName** The name of the core group whose core group access points will be deleted. (String required) • Returns: None	**Batch mode example usage:** • Using Jacl: $AdminTask deleteCoreGroupAccessPoints (cells/ rohitbuildCell01 |coregroupbridge.xml# CoreGroupBridgeSettings_1) "-coreGroupName *DefaultCoreGroup*" • Using Jython: AdminTask .deleteCoreGroupAccessPoints ('(cells/rohitbuildCell01 |coregroupbridge.xml #CoreGroupBridgeSettings_1)', '[-coreGroupName *DefaultCoreGroup*]') **Interactive mode example usage:** • Using Jacl: $AdminTask deleteCoreGroupAccessPoints {-interactive} • Using Jython: AdminTask .deleteCoreGroupAccessPoints ('[-interactive]')
delete SIB Destination	SIB Admin Commands	Use the **delete SIB Destination** command to delete a bus destination. This command deletes the named destination of the named bus and deletes all related message points.	None	• Parameters: **bus** name of the bus on which the destination to be deleted exists (String, required) **name** name of the destination to be deleted (String, required) • Returns: None	**Batch mode example usage:** • Using Jacl: $AdminTask deleteSIBDestination {-bus *busname* -name *destname*} • Using Jython: AdminTask .deleteSIBDestination ('[-bus *busname* -name *destname*]') **Interactive mode example usage:** • Using Jacl: $AdminTask deleteSIBDestination {-interactive} • Using Jython: AdminTask .deleteSIBDestination ('[-interactive]')

delete SIB Engine	SIB Admin Commands	Use the **delete SIB Engine** command to delete the default or named bus messaging engine from the named SIB bus. A server can only have one messaging engine, so when using this command to delete a messaging engine from a server there is no need to supply the engine name. A cluster can have more than one messaging engine so the name of the engine must be supplied.	None	- Parameters:
				***bus** name of the bus to which the messaging engine to be deleted belongs (String, required)
				node to delete a messaging engine on a server, supply node and server name, but not cluster name (String, optional)
				server to delete a messaging engine on a server, supply node and server name, but not cluster name (String, optional)
				cluster to delete a messaging engine on a cluster, supply cluster name, but not node and server name (String, optional)
				engine name of the messaging engine to delete. This is optional, and is only required when deleting a messaging engine from a cluster (String, optional)
				- Returns: None
					Batch mode example usage:
					- Using Jacl:
					`$AdminTask deleteSIBEngine {-bus *busname* -node *nodeName* -server *severname*}`
					- Using Jython:
					`AdminTask.deleteSIBEngine ('[-bus *busname* -node *nodeName* -server *severname*]')`
					Interactive mode example usage:
					- Using Jacl:
					`$AdminTask deleteSIBEngine {-interactive}`
					- Using Jython:
					`AdminTask .deleteSIBEngine ('[-interactive]')`

delete SIB JMS Activation Spec	SIB JMS Admin Commands	Use the **delete SIB JMS Activation Spec** command to delete an activation specification.	None	- Parameters:
				name The name of the activation specification that you want to delete. (String, (required)
				- Returns: None
					Batch mode example usage:
					- Using Jacl:
					`$AdminTask deleteSIBJMSActivationSpec {-name *specname*}`
					- Using Jython:
					`AdminTask .deleteSIBJMSActivationSpec ('[-name *specname*]')`
					Interactive mode example usage:
					- Using Jacl:
					`$AdminTask deleteSIBJMSActivationSpec {-interactive}`
					- Using Jython:
					`AdminTask .deleteSIBJMSActivationSpec ('[-interactive]')`

delete SIB JMS Connection Factory	SIB JMS Admin Commands	Use the **delete SIB JMS Connection Factory** command to	None	- Parameters:
				name The name of the SIB JMS connection factory (String, required)
				- Returns: None
					Batch mode example usage:
					- Using Jacl:
					`$AdminTask deleteSIBJMSConnectionFactory {-name *factory_name*}`
					- Using Jython:
					`AdminTask .deleteSIBJMSConnectionFactory ('[-name *factory_name*]')`
					Interactive mode example usage:
					- Using Jacl:
					`$AdminTask deleteSIBJMSConnectionFactory {-interactive}`
					- Using Jython:
					`AdminTask .deleteSIBJMSConnectionFactory ('[-interactive]')`

delete SIB JMS Queue	SIB JMS Admin Commands	Use the **delete SIB JMS Queue** command to delete a JMS queue.	None	• Parameters:
				name
				The name of the SIB JMS queue. (String, required)
				• Returns: None
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask deleteSIBJMSQueue {-name queue_name}`
					• Using Jython:
					`AdminTask .deleteSIBJMSQueue ('[-name queue_name]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask deleteSIBJMSQueue {-interactive}`
					• Using Jython:
					`AdminTask .deleteSIBJMSQueue ('[-interactive]')`

delete SIB JMS Topic	SIB JMS Admin Commands	Use the **delete SIB JMS Topic** command to delete a JMS topic.	None	• Parameters:
				name
				The name of the SIB JMS topic (String, required)
				• Returns: None
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask deleteSIBJMSTopic {-name topic_name}`
					• Using Jython:
					`AdminTask .deleteSIBJMSTopic ('[-name topic_name]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask deleteSIBJMSTopic {-interactive}`
					• Using Jython:
					`AdminTask .deleteSIBJMSTopic ('[-interactive]')`

delete SIB Mediation	SIB Admin Commands	Use this command to delete the named mediation from the named bus.	None	• Parameters:
				bus
				name of the bus that owns the mediation (String, required)
				mediation Name
				name of the mediation to be deleted (String, required)
				• Returns: None
					Batch mode example usage:
					• Using Jacl:
					• Using Jython:
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask deleteSIBMediation {-interactive}`
					• Using Jython:
					`AdminTask .deleteSIBMediation ('[-interactive]')`

delete SIBWS EndpointListener	SIB Web Services group	The **delete SIBWS Endpoint Listener** command deletes the configuration oa an end point listener. This command fails if there are inbound port objects associated with the end point listener.	Object name of the end point listener that you want to delete.	• Parameters: None
				• Returns: None
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask deleteSIBWSEndpointListener $epl`
					• Using Jython:
					`AdminTask .deleteSIBWSEndpointListener (epl)`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask deleteSIBWSEndpointListener {-interactive}`
					• Using Jython:
					`AdminTask .deleteSIBWSEndpointListener ('[-interactive]')`

delete SIBWS Inbound Service	SIB Web Services group	The **delete SIBWS Inbound Service** command deletes an inbound service object and any inbound port objects that are associated.	The object name of the inbound service object that you want to delete.	• Parameters:
				userId The user ID to use to interact with UDDI registries. (optional)
				password The password to use to interact with UDDI registries. (optional)
				• Returns: None
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask deleteSIBWSInboundService $inService`
					• Using Jython:
					`AdminTask .deleteSIBWSInboundService (inService)`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask deleteSIBWSInboundService {-interactive}`
					• Using Jython:
					`AdminTask .deleteSIBWSInboundService ('[-interactive]')`

delete SIBWS Outbound Service	SIB Web Services group	The **delete SIBWS Outbound Service** command deletes an outbound service object and any outbound port objects that are associated. Resources that are associated with the outbound service or outbound ports, for example, WS-Security configuration, are disassociated from the outbound service and the outbound ports but are not deleted.	Object name of the outbound service object that you want to delete.	• Parameters: None
				• Returns: None
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask deleteSIBWSOutboundService $outService`
					• Using Jython:
					`AdminTask .deleteSIBWSOutboundService (outService)`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask deleteSIBWSOutboundService {-interactive}`
					• Using Jython:
					`AdminTask .deleteSIBWSOutboundService ('[-interactive]')`

deleteSIBus	SIB Admin Commands	Use this command to delete the named SIB bus. Also deletes all SIB mediations and SIB destinations owned by the bus.	None	- Parameters:
				bus
				name of bus to be deleted from the current cell (String, required)
				- Returns: None
					Batch mode example usage:
					- Using Jacl:
					`$AdminTask deleteSIBus {-bus bus_name}`
					- Using Jython:
					`AdminTask.deleteSIBus ('[-bus bus_name]')`
					Interactive mode example usage:
					- Using Jacl:
					`$AdminTask deleteSIBus {-interactive}`
					- Using Jython:
					`AdminTask.deleteSIBus ('[-interactive]')`

deleteServer	None	Use the **deleteServer** command to delete the server scope configuration and the server entry that corresponds to it in the `server index.xml` file. You can also use this command to delete a Web server.	None	- Parameters:
				-nodeName
				(String, required)
				-serverName
				(String, required)
				- Returns: None
					Batch mode example usage:
					- Using Jacl:
					`$AdminTask deleteServer {-nodeName node_name -serverName server_name}`
					- Using Jython:
					`AdminTask.deleteServer ('[-nodeName node_name -serverName server_name]')`
					Interactive mode example usage:
					- Using Jacl:
					`$AdminTask deleteServer {-interactive}`
					- Using Jython:
					`AdminTask.deleteServer ('[-interactive]')`

delete Server Template	None	Use the **delete Server Template** command to delete server templates. You cannot delete templates defined by the system. You can only delete server templates that you created. This command deletes the directory that hosts the server template.	A server template identification, `javax.management.ObjectName`. This target object is required.	• Returns: Void	**Interactive mode example usage:**
					• Using Jacl:
					`$AdminTask deleteServerTemplate {-interactive}`
					• Using Jython:
					`AdminTask.deleteServerTemplate('[-interactive]')`

delete WSGW Gateway Service	WS Gateway group	The **delete WSGW Gateway Service** command deletes a gateway service. It deletes the gateway destination the corresponding reply destination, inbound service, and inbound port enablement objects, and all of the target service objects that are associated. This command does not delete the destinations that are associated with the target services.	Object name of the gateway service object	- Parameters: None
				- Returns: None
					Batch mode example usage:
					- Using Jacl:
					`$AdminTask deleteWSGWGatewayService $gwService`
					- Using Jython:
					`AdminTask .deleteWSGWGatewayService (gwService)`
					Interactive mode example usage:
					- Using Jacl:
					`$AdminTask deleteWSGWGatewayService {-interactive}`
					- Using Jython:
					`AdminTask .deleteWSGWGatewayService ('[-interactive]')`

delete WSGW Proxy Service	WS Gateway group	The **delete WSGW Proxy Service** command deletes a proxy service including the proxy destinations, outbound service, outbound ports, inbound service, and inbound port enablement objects.	Object name of the Proxy Service object	- Parameters: None
				- Returns: None
					Batch mode example usage:
					- Using Jacl:
					`$AdminTask deleteWSGWProxyService $proxyService`
					- Using Jython:
					`AdminTask .deleteWSGWProxyService (proxyService)`
					Interactive mode example usage:
					- Using Jacl:
					`$AdminTask deleteWSGWProxyService {-interactive}`
					- Using Jython:
					`AdminTask .deleteWSGWProxyService ('[-interactive]')`

disconnect SIBWS Endpoint Listener	SIB Web Services group	The **disconnect SIBWS Endpoint Listener** command disconnects an end point listener from a bus.	Object name of the end point listener to be disconnected.	• Parameters: **bus** The name of the bus from which to be disconnected. (required) • Returns: None	**Batch mode example usage:** • Using Jacl: $AdminTask disconnectSIBWS EndpointListener $epl {-bus "*MyBus*"} • Using Jython: AdminTask .disconnectSIBWS EndpointListener (epl,'[-bus *MyBus*]') **Interactive mode example usage:** • Using Jacl: $AdminTask disconnectSIBWS EndpointListener {-interactive} • Using Jython: AdminTask .disconnectSIBWS EndpointListener ('[-interactive]')
does Core Group Exist	Core Group Managemen group	The **does Core Group Exist** command returns a boolean value that indicates if the core group that you specify exists.	None	• Parameters: **coreGroupName** The name of the core group. (String, required) • Returns: A boolean value.	**Batch mode example usage:** • Using Jacl: $AdminTask doesCoreGroupExist {-coreGroupName *MyCoreGroup*} • Using Jython: AdminTask .doesCoreGroupExist ('[-coreGroupName *MyCoreGroup*]') **Interactive mode example usage:** • Using Jacl: $AdminTask doesCoreGroupExist {-interactive} • Using Jython: AdminTask .doesCoreGroupExist ('[-interactive]')

export Server	Configuration archive Operations group	Use the **export Server** command to export the server configuration to a node defined in the configuration archive.
		The **export Server** command virtualizes the server configuration and exports a server to a configuration archive. This process breaks any existing associations between the server configurations in the configuration archive and the configurations in the system. This process also removes applications from the server that you specify, breaks the relationship between the server that you specify and the core group of the server, cluster, or SIBus membership. The **exportServer** command exports the metadata file of the node where the server resides. You can use this information
			None	• Parameters:
				-archive
				The fully qualified path of the exported configuration archive. (String, required)
				-nodeName
				The node name of the server. This parameter is only required when the server name is not unique across the cell. (String, optional)
				-serverName
				The server name. (String, required)
				• Returns: None
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask exportServer {-archive c:\myServer.ear -nodeName node1 -serverName server1}`
					• Using Jython:
					`AdminTask.exportServer ('[-archive c:\myServer.ear -nodeName node1 -serverName server1]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask exportServer {-interactive}`
					• Using Jython:
					`AdminTask.exportServer ('[-interactive]')`

export Was profile	configuration archive Operations group	Use the **export Was profile** command to export the entire cell configuration to a configuration archive.	None	- Parameters:
				archive
				The fully qualified file path of the exported configuration archive. (String, required)
				- Returns: None
					Batch mode example usage:
					- Using Jacl:
					`$AdminTask exportWasprofile {-archive c:\myCell.ear}`
					- Using Jython:
					`AdminTask .exportWasprofile ('[-archive c:\myCell.ear]')`
					Interactive mode example usage:
					- Using Jacl:
					`$AdminTask exportWasprofile {-interactive}`
					- Using Jython:
					`AdminTask .exportWasprofile ('[-interactive]')`

get All Core Group ames	Core Group Management group	The **get All Core Group Names** command returns a string that contains the names of all of the existing core groups	None	- Parameters: None
				- Returns: String array (String[ ])
					Batch mode example usage:
					- Using Jacl:
					`$AdminTask getAllCoreGroupNames`
					- Using Jython:
					`AdminTask .getAllCoreGroupNames()`
					Interactive mode example usage:
					- Using Jacl:
					`$AdminTask getAllCoreGroupNames {-interactive}`
					- Using Jython:
					`AdminTask .getAllCoreGroupNames ('[-interactive]')`

get Core Group Name For Server	Core Group Management group	The **get Core Group Name For Server** command returns the name of the core group for which the server you specify is currently a member.	None	• Parameters:
				- nodeName The name of the node that contains the server. (String, required)
				- serverName The name of the server. (String, required)
				• Returns: The name of the core group that currently contains the server that you specified. (String)
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask getCoreGroupNameForServer {-nodeName` *`myNode`* `-serverName` *`myServer`*`}`
					• Using Jython:
					`AdminTask .getCoreGroupNameForServer ('[-nodeName` *`myNode`* `-serverName` *`myServer`*`]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask getCoreGroupNameForServer {-interactive}`
					• Using Jython:
					`AdminTask .getCoreGroupNameForServer ('[-interactive]')`

get Default Core Group Name	Core Group Management group	The **get Default Core Group Name** command returns the name of the default core group.	None	• Parameters: None
				• Returns: String
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask getDefaultCoreGroupName`
					• Using Jython:
					`AdminTask .getDefaultCoreGroupName()`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask getDefaultCoreGroupName {-interactive}`
					• Using Jython:
					`AdminTask .getDefaultCoreGroupName ('[-interactive]')`

get Metadata Properties	Managed Object Metadata group	The **get Metadata Properties** command obtains all metadata for the node that you specify.	None	• Parameters:
				- nodeName
				The name of the node associated with the metadata you want this command to return.
				• Returns: The list of metadata properties.
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask getMetadataProperties {-nodeName node1}`
					• Using Jython:
					`AdminTask .getMetadataProperties ('[-nodeName node1]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask getMetadataProperties {-interactive}`
					• Using Jython:
					`AdminTask .getMetadataProperties ('[-interactive]')`

get Metadata Property	Managed Object Metadata group	The **get Metadata Property** command obtains metadata with the specified key for the node that you specify.	None	• Parameters:
				- nodeName
				The name of the node associated with the metadata you want this command to return.
				- propertyName
				Metadata property key.
				• Returns: The requested property for the node that you specified.
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask getMetadataProperty {-nodeName node1 -propertyName com.ibm.websphere .baseProductVersion}`
					• Using Jython:
					`AdminTask .getMetadataProperty ('[-nodeName node1 -propertyName com.ibm.websphere .baseProductVersion]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask getMetadataProperty {-interactive}`
					• Using Jython:
					`AdminTask .getMetadataProperty ('[-interactive]')`

get Named TCP End Point	Core Group Bridge Management group	The **get Named TCP End Point** command returns the port associated with the bridge interface that you specify. The port that is returned is the one that is specified on the TCP inbound channel of the transport channel chain for bridge interface that you specify.	The bridge interface object for which the port will be listed. (Object Name, required)	• Parameters: None
				• Returns: The port (named end point) object name of the TCP inbound channel instance which resides on the DCS transport channel chain of the bridge interface.
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask getNamedTCPEndPoint (cells/rohitbuildCell01|coregroupbridge.xml #BridgeInterface_2)`
					• Using Jython:
					`AdminTask .getNamedTCPEndPoint ('(cells/rohitbuildCell01 |coregroupbridge.xml #BridgeInterface_2)')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask getNamedTCPEndPoint {-interactive}`
					• Using Jython:
					`AdminTask .getNamedTCPEndPoint ('[-interactive]')`

get Node Base Product Version	Managed Object Metadata group	The **get Node Base Product Version** command returns the version of the WebSphere Application Server for a node that you specify. This command only returns the version for a distributed installation of the product.	None	• Parameters:
				- nodeName
				The name of the node associated with the metadata you want this command to return.
				• Returns: WebSphere Application Server version for the node that you specify.
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask getNodeBaseProductVersion {-nodeName` *node1*`}`
					• Using Jython:
					`AdminTask .getNodeBaseProductVersion ('[-nodeName` *node1*`]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask getNodeBaseProductVersion {-interactive}`
					• Using Jython:
					`AdminTask .getNodeBaseProductVersion ('[-interactive]')`

get Node Major Version	Managed Object Metadata group	The **get Node Major Version** command returns the major version of the WebSphere Application Server for a node that you specify. It only returns the version for a distributed installation of the product.	None	• Parameters:
				nodeName The name of the node associated with the metadata you want this command to return.
				• Returns: WebSphere Application Server major version for the node that you specified.
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask getNodeMajorVersion {-nodeName *node1*}`
					• Using Jython:
					`AdminTask .getNodeMajorVersion ('[-nodeName *node1*]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask getNodeMajorVersion {-interactive}`
					• Using Jython:
					`AdminTask .getNodeMajorVersion ('[-interactive]')`

get Node Minor Version	Managed Object Metadata group		None	• Parameters:
				- nodeName The name of the node associated with the metadata you want this command to return.
				• Returns: WebSphere Application Server minor version for the node that you specified.
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask getNodeMinorVersion {-nodeName *node1*}`
					• Using Jython:
					`AdminTask .getNodeMinorVersion ('[-nodeName *node1*]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask getNodeMinorVersion {-interactive}`
					• Using Jython:
					`AdminTask .getNodeMinorVersion ('[-interactive]')`

get Node Platform OS	Managed Object Metadata group	The **get Node Platform OS** command returns the operating system name for a node that you specify.	None	• Parameters:
				- nodeName The name of the node associated with the metadata you want this command to return.
				• Returns: The operating system name of the node that you specified.
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask getNodePlatformOS {-nodeName node1}`
					• Using Jython:
					`AdminTask .getNodePlatformOS ('[-nodeName node1]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask getNodePlatformOS {-interactive}`
					• Using Jython:
					`AdminTask .getNodePlatformOS ('[-interactive]')`

get Node Sysplex Name	Managed Object Metadata group	The **get Node Sysplex Name** command returns the sysplex name for a node that you specify.	None	• Parameters:
				- nodeName The name of the node associated with the metadata you want this command to return.
				• Returns: The sysplex name of the given node.
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask getNodeSysplexName {-nodeName node1}`
					• Using Jython:
					`AdminTask .getNodeSysplexName ('[-nodeName node1]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask getNodeSysplexName {-interactive}`
					• Using Jython:
					`AdminTask .getNodeSysplexName ('[-interactive]')`

get Server Type					**Interactive mode example usage:**
					• Using Jacl:
					`$AdminTask getServerType {-interactive}`
					• Using Jython:
					`AdminTask.getServerType ('[-interactive]')`

get TCP End Point	None	The **get TCP End Point** command obtains the named end point associated with either a TCP inbound channel or a chain that contains a TCP inbound channel.	TCP Inbound Channel, or containing chain, instance that is associated with a Named EndPoint. (Object Name, required)	• Parameters: None
				• Returns: Object name of an existing named end point that is associated with the TCP inbound channel instance or a channel chain.
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask getTCPEndPoint`
					`TCP_1(cells`
					`/rohitbuildCell01/`
					`nodes/rohitbuildCellManager01`
					`/servers/dmgr|server.xml`
					`#TCPInboundChannel_1)`
					`$AdminTask getTCPEndPoint`
					`DCS`
					`(cells/rohitbuildCell01`
					`/nodes/`
					`rohitbuildCellManager01`
					`/servers`
					`/dmgr|server.xml#Chain_3)`
					• Using Jython:
					`AdminTask.getTCPEndPoint`
					`('TCP_1(cells/rohitbuildCell01/`
					`nodes/rohitbuildCellManager01`
					`/servers/dmgr`
					`|server.xml`
					`#TCPInboundChannel_1)')`
					`AdminTask.getTCPEndPoint`
					`('DCS(cells/rohitbuildCell01`
					`/nodes/`
					`rohitbuildCellManager01`
					`/servers`
					`/dmgr|server.xml#Chain_3)')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask getTCPEndPoint`
					`{-interactive}`
					• Using Jython:
					`AdminTask.getTCPEndPoint`
					`('[-interactive]')`

import Server	Configuration archive Operations group	Use the **import Server** command to import a server that resides in a configuration archive to the system. This command imports all the server scope configurations defined in the configuration archive to system configuration.	None	• Parameters:
				-archive
				The fully qualified path of the configuration archive. (String, required)
				-node InArchive
				The node name of the server defined in the configuration archive. (String, optional if there is only one node defined in the configuration archive, required if there are multiple nodes defined in the configuration archive)
				-server InArchive
				The name of the server defined in the configuration archive. (String, optional if there is only one server defined on the specified *nodeIn Configuration* archive, required if there are multiple servers defined under the specified *nodeIn Configuration* archive)
				-nodeName
				The node name where the server is imported. (String, optional if there is only one node)
				-serverName
				The server name where the server is imported. If the server name that you specify matches an existing server name under the node, an exception is thrown. (String, optional, default:serve
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask importServer {-archive c:\myServer.ear -nodeInArchive node1 -serverInArchive server1}`
					• Using Jython:
					`AdminTask.importServer ('[-archive c:\myServer.ear -nodeInArchive node1 -serverInArchive server1]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask importServer {-interactive}`
					• Using Jython:
					`AdminTask.importServer ('[-interactive]')`

import Server continued				• Parameters continued: **-serverName** The server name where the server is imported. If the server name that you specify matches an existing server name under the node, an exception is thrown. (String, optional, default:serverInArchive) **-coreGroup** The core group name to which the server should belong. (String, optional) • Returns: None	

help	None	The **help** command provides a summary of the help commands and ways to invoke an administrative command.	None	• Parameters: None
				• Returns: A general help description
					• Using Jacl:
					`$AdminTask help`
					• Using Jython:
					`print AdminTask.help()`

help	None	The **help** command provides a list of available administrative commands if the option string is `-commands` or administrative command groups if the option string is `-commandGroups`. Valid options include `-commands` and `-commandGroups`.	None	• Parameters:
				- options
				• Returns: A summary of all available administrative commands.
					• Using Jacl:
					`$AdminTask help -commands`
					• Using Jython:
					`AdminTask.help('-commands')`

help	None	If you provide the step name, this command provides help information for a given step of an administrative command. Otherwise, it provides help information for a given admin command or administrative command group. The stepName parameter is optional.	None	• Parameters:
				– - commandName
				– - stepName
				• Returns: A summary of the specified command group, administrative command, or step.
					• Using Jacl:
					`$AdminTask help createJ2CConnectionFactory`
					• Using Jython:
					`AdminTask.help ('createJ2CConnectionFactory')`

import Was profile	configuration archive Operations group	Use the **import Was profile** command to import a cell configuration in the configuration archive to the system. Only a base single server configuration is supported for this command.	None	• Parameters:
				archive
				The fully qualified file path of the configuration archive. (String, required)
				• Returns: Void
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask importWasprofile {-archive c:\myCell.ear}`
					• Using Jython:
					`AdminTask.importWasprofile ('[-archive c:\myCell.ear]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask importWasprofile {-interactive}`
					• Using Jython:
					`AdminTask.importWasprofile ('[-interactive]')`

is Node ZOS	Managed Object Metadata group	The **is Node ZOS** command tests if a node that you specify is running on the z/OS platform. This command does not apply to distributed platforms or WebSphere Application Server-Express.	None	- Parameters:
				- nodeName
				The name of the node associated with the metadata you want this command to return.
				- Returns: A `true` value if the node operating system is z/OS. A `false` value if the node operating system is not z/OS.
					Batch mode example usage:
					- Using Jacl:
					`$AdminTask isNodeZOS {-nodeName` *`node1`*`}`
					- Using Jython:
					`AdminTask.isNodeZOS ('[-nodeName` *`node1`*`]')`
					Interactive mode example usage:
					- Using Jacl:
					`$AdminTask isNodeZOS {-interactive}`
					- Using Jython:
					`AdminTask.isNodeZOS ('[-interactive]')`

list Admin Object Interfaces	JCA management group	Use the **list Admin Object Interfaces** command to list the administrative object interfaces defined under the resource adapter that you specify.	J2C Resouce adapter object ID	- Parameters: None
				- Returns: A list of administrative object interfaces.
					Batch mode example usage:
					- Using Jacl:
					`$AdminTask listAdminObjectInterfaces $ra`
					- Using Jython:
					`AdminTask .listAdminObjectInterfaces (ra)`

list Chain Templates	Channel Framework Management group	The **list Chain Templates** command displays a list of templates that you can use to create chains in this configuration. All templates have a certain type of transport channel as the last transport channel in the chain.	None	• Parameters:
				- acceptorFilter The templates returned by this method all have a transport channel instance of the specified type as the last transport channel in the chain. (String, optional)
				• Returns: A list of all the chain template object names. If you specify the acceptorFilter parameter, the list that returns is filtered to match the filter that you specified.
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask listChainTemplates {}`
					`$AdminTask listChainTemplates "-acceptorFilter` *`WebContainerInboundChannel`*`"`
					• Using Jython:
					`AdminTask .listChainTemplates()`
					`AdminTask .listChainTemplates ('[-acceptorFilter` *`WebContainerInboundChannel`*`]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask listChainTemplates {-interactive}`
					• Using Jython:
					`AdminTask .listChainTemplates ('[-interactive]')`

list Chains	Channel Framework Management group	The **list Chains** command lists all the chains configured under a particular instance of the transport channel service.	The instance of the transport channel service under which the chains are configured. (Object Name, required)	• Parameters:
				- acceptorFilter
				The chains that are returned by this parameter will have a transport channel instance of the type that you specify as the last transport channel in the chain. (String, optional)
				- endPoint Filter:
				The chains returned by this parameter will have a TCP inbound channel using an end point with the name that you specify.(String, optional)
				• Returns: A list of all the channel chain object names that match the specified filters. If no you do not specify any parameters, all of the channel chains that are configured under the particular instance of transport channel service are returned.
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask listChains (cells/rohitbuildCell01/nodes/rohitbuildNode01/servers/server2|server.xml#TransportChannelService_1093445762328)`
					`$AdminTask listChains (cells/rohitbuildCell01/nodes/rohitbuildNode01/servers/server2|server.xml#TransportChannelService_1093445762328) {-acceptorFilter` *`WebContainerInboundChannel`*`}`
					`$AdminTask listChains (cells/rohitbuildCell01/nodes/rohitbuildNode01/servers/server2|server.xml#TransportChannelService_1093445762328) {-endPointFilter` *`WC_adminhost`*`}`
					• Using Jython:
					`AdminTask.listChains('(cells/rohitbuildCell01/nodes/rohitbuildNode01/servers/server2|server.xml#TransportChannelService_1093445762328)')`
					`AdminTask.listChains('(cells/rohitbuildCell01/nodes/rohitbuildNode01/servers/server2|server.xml#TransportChannelService_1093445762328)', '[-acceptorFilter WebContainerInboundChannel]')`
					`AdminTask.listChains('(cells/rohitbuildCell01/nodes/rohitbuildNode01/servers/server2|server.xml#TransportChannelService_1093445762328)', '[-endPointFilter WC_adminhost]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask listChains {-interactive}`
					• Using Jython:
					`AdminTask.listChains ('[-interactive]')`

list Connection Factory Interfaces	JCA management group	Use the **list Connection Factory Interfaces** command to list all of the connection factory interfaces defined under the Java 2 resource adapter that you specify.	J2C Resource Adapter object ID	• Parameters: None
				• Returns: A list of connection factory interfaces.
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask listConnectionFactory Interfaces $ra`
					• Using Jython:
					`AdminTask .listConnectionFactory Interfaces (ra)`

list Core Groups	Core Group Bridge Management group	The **list Core Groups** command returns a collection of core groups that are related to the core group that you specify.	The name of the core group for which the related core groups will be listed. (String, required)	• Parameters:
				- cgBridge Settings The group bridge settings object for the cell. (Object Name, required)
				• Returns: A set of core group names.
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask listCoreGroups DefaultCoreGroup "-cgBridgeSettings (cells/rohitbuildCell01 |coregroupbridge.xml# CoreGroupBridgeSettings_1)"`
					• Using Jython:
					`AdminTask.listCoreGroups ('DefaultCoreGroup', '[-cgBridgeSetting (cells/rohitbuildCell01 |coregroupbridge.xml #CoreGroupBridgeSettings_1)]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask listCoreGroups {-interactive}`
					• Using Jython:
					`AdminTask.listCoreGroups ('[-interactive]')`

list Eligible Bridge Interfaces	Core Group Bridge Management group	The **list Eligible Bridge Interfaces** command returns a collection of node, server and transport channel chain combinations that are eligible to become bridge interfaces for the specified core group access point.	The core group access point object for which bridge interfaces will be listed. (Object Name, required)	• Parameters: None
				• Returns: A set of bridge interfaces. (Set of String) Each bridge interface is represented by a combination of a node, a server and a DCS channel chain: <node *name*>, <server *name*>, <DCS Channel Chain *objectName*. For example, an element of the set returned by this command may look like the following: `rohitbuild dmgr DCS-Secure (cells/rohitbuildCell/nodes/rohitbuild/servers/dmgr|server.xml#Chain_4)`
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask listEligibleBridgeInterfaces CGAP_DCG_2 (cells/rohitbuildCell01|coregroupbridge.xml#CoreGroupAccessPoint_1089636614062)`
					• Using Jython:
					`AdminTask.listEligibleBridgeInterfaces('CGAP_DCG_2 (cells/rohitbuildCell01|coregroupbridge.xml#CoreGroupAccessPoint_1089636614062)')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask listEligibleBridgeInterfaces {-interactive}`
					• Using Jython:
					`AdminTask.listEligibleBridgeInterfaces('[-interactive]')`

list J2C Activation Specs	JCA management group	Use the **list J2c Activation Specs** command to list the activation specs contained under the resource adapter and message listener type that you specify.	J2C Resource Adapter object ID	• Parameters:
				-message ListenerType Specifies the message listener type for the resource adapter for which you are making a list. This parameter is required.
				• Returns: A list of activation specs that has specified message Listener type.
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask listJ2CActivationSpecs $ra {-messageListenerType javax.jms.MessageListener}`
					• Using Jython:
					`AdminTask.listJ2CActivationSpecs(ra, '[-messageListenerType javax.jms.MessageListener]')`

list J2C Admin Objects	JCA Management group	Use the **list J2C Admin Objects** command to list administrative objects that contains the administrative object interface that you specify.	J2C Resource Adapter object ID	• Parameters:
				-adminObject Interface
				Specifies the administrative object interface for which you want to list. This parameter is required.
				• Returns: A list of administrative objects that has specified adminObject Interface.
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask listJ2CAdminObjects $ra {-adminObjectInterface fvt.adaptor.message .FVTMessageProvider}`
					• Using Jython:
					`AdminTask .listJ2CAdminObjects (ra, '[-adminObjectInterface fvt.adaptor.message .FVTMessageProvider]')`

list J2C Connection Factories	JCA management group	Use the **list J2C Connection Factories** command to list the Java 2 connection factories under the resource adapter and connection factory interface that you specify	J2C Resource Adapter object ID	• Parameters:
				-connection Factory Interface
				Indicates the name of the connection factory that you want to list. This parameter is required.
				• Returns: A list of J2C connection Factory that has the specified connection Factory Interface.
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask listJ2CConnectionFactories $ra {-connectionFactoryInterface javax.sql.DataSource}`
					• Using Jython:
					`AdminTask .listJ2CConnectionFactories (ra, '[-connectionFactoryInterface javax.sql.DataSource]')`

list Managed Nodes	Unmanaged Node Commands group	Use the **list Managed Nodes** command to list the managed nodes (nodes that have a node agent defined) in a configuration.	None	• Parameters: None
				• Returns: List
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask listManagedNodes`
					• Using Jython:
					`AdminTask.listManagedNodes()`

list Message Listener Types	JCA Management group	Use the **list Message Listener Types** command to list the message listener types defined under the resource adapter that you specify.	J2C Resource Adapter object ID	• Parameters: None
				• Returns: A list of message listener types.
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask listMessageListenerTypes $ra`
					• Using Jython:
					`AdminTask .listMessageListenerTypes (ra)`

list Node Group Properties	Node Group Commands group	The **list Node Group Properties** command displays all of the custom properties of a node group.	The target object is name of the node group. This target object is required.	- Parameters: None
				- Returns: A list of all of the custom properties of a node group.
					Batch mode example usage:
					- Using Jacl:
					`$AdminTask listNodeGroupProperties WBINodeGroup`
					- Using Jython:
					`AdminTask .listNodeGroupProperties ('WBINodeGroup')`
					Interactive mode example usage:
					- Using Jacl:
					`$AdminTask listNodeGroupProperties {-interactive}`
					- Using Jython:
					`AdminTask .listNodeGroupProperties ('[-interactive]')`

list Node Groups	Node Group Commands group	The **list Node Groups** command returns the list of node groups from the configuration repository. You can pass an optional node name to the command that returns the list of node groups where the node resides.	The target object is name of the node. This target object is optional.	- Parameters: None
				- Returns: A list of the node groups in the cell.
					Batch mode example usage:
					- Using Jacl:
					`$AdminTask listNodeGroups`
					`$AdminTask listNodeGroups` *`nodeName`*
					- Using Jython:
					`AdminTask.listNodeGroups`
					`AdminTask.listNodeGroups ('`*`nodeName`*`')`
					Interactive mode example usage:
					- Using Jacl:
					`$AdminTask listNodeGroups {-interactive}`
					- Using Jython:
					`AdminTask.listNodeGroups ('[-interactive]')`

list Nodes	Node Group Commands group	The **list Nodes** command displays all of the nodes in the cell.	The target object is name of the node group. This target object is optional.	• Parameters: None
				• Returns: A list of all the nodes in the cell
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask listNodes`
					• Using Jython:
					`AdminTask.listNodes()`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask listNodes {-interactive}`
					• Using Jython:
					`AdminTask.listNodes ('[-interactive]')`

list SIB Destinations	SIB Admin Commands	Use this command to get a list of SIB destinations of the named type owned by a named SIB bus. If no type is named, all destinations owned by the named bus are listed.	None	• Parameters:
				bus
				Bus name (String, required)
				name
				Destination name (String, required)
				type
				type of destination to list - Queue, TopicSpace, WebService or Port (String, optional)
				• Returns: List of SIB destinations.
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask listSIBDestinations {-bus` *`busname`* `-name` *`destname`* `-type Queue}`
					• Using Jython:
					`AdminTask .listSIBDestinations ('[-bus` *`busname`* `-name` *`destname`* `-type Queue]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask listSIBDestinations {-interactive}`
					• Using Jython:
					`AdminTask .listSIBDestinations ('[-interactive]')`

list SIB Engines	SIB Admin Commands	Use the **list SIB Engines** command to get a list of bus messaging engines. Supplying only the bus parameter will result in a list of all engines associated with the named bus. Supplying only the node and server parameters will result in a list of all engines owned by the named node/server. Supplying only the cluster parameter will result in a list of all engines owned by the named cluster. All other parameter combinations are illegal.	None	• Parameters:
				bus
				name of the bus whose engines are to be listed (String, optional)
				node
				node name. To list messaging engines on a server, supply node and server name, but not cluster name (String, optional)
				server
				server name. To list messaging engines on a server, supply node and server name, but not cluster name (String, optional)
				cluster
				cluster name. To list messaging engines on a cluster, supply cluster name, but not node and server name (String, optional)
				• Returns: A list of SIB messaging engines.
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask`
					`listSIBEngines {-bus`
					` `*`busname`*` -node`
					`nodeName`
					` -server `*`severname`*`}`
					• Using Jython:
					`AdminTask.listSIBEngines`
					`('[-bus`
					` `*`busname`*` -node`
					`nodeName`
					` -server `*`severname`*`]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask listSIBEngines`
					`{-interactive}`
					• Using Jython:
					`AdminTask.listSIBEngines`
					`('[-interactive]')`

list SIB JMS Activation Specs	SIB JMS Admin Commands				**Interactive mode example usage:**
					• Using Jacl:
					`$AdminTask`
					` listSIBJMSActivationSpecs`
					` {-interactive}`
					• Using Jython:
					`AdminTask`
					`.listSIBJMSActivationSpecs`
					` ('[-interactive]')`

list SIB JMS Connection Factories	SIB JMS Admin Commands	Use the **list SIB JMS Connection Factories** command to list all of the JMS connection factories for the default messaging provider at the scope that you specify.	None	• Parameters:
				type
				Filters the list of connection factories. Valid values include:
				– all - Lists all the JMS connection factories (unified, queue, and topic) at the scope that you specify.
				– queue - Lists all of the JMS queue connection factories at the scope that you specify.
				– topic - Lists all of the JMS topic connection factories at the scope that you specify.
				If you do no specify the type option, this command will return only the unified JMS connection factories at the scope that you specified.
				• Returns: A list of connection factories at the scope that you specified.
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask listSIBJMSConnectionFactories {-type queue}`
					• Using Jython:
					`AdminTask.listSIBJMSConnectionFactories('[-type queue]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask listSIBJMSConnectionFactories {-interactive}`
					• Using Jython:
					`AdminTask.listSIBJMSConnectionFactories('[-interactive]')`

list SIB JMS Queues	SIB JMS Admin Commands	Use the **list SIB JMS Queues** command to list all the JMS queues for the default messaging provider at the specified scope.	None	• Parameters: None
				• Returns: None
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask listSIBJMSQueues`
					• Using Jython:
					`AdminTask.listSIBJMSQueues()`

list SIB JMS Topics	SIB JMS Admin Commands	Lists all JMS topics for the default messaging provider at the specified scope.	None	• Parameters: None
				• Returns: A list of JMS topics.
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask listSIBJMSTopics`
					• Using Jython:
					`AdminTask.listSIBJMSTopics()`

list SIB Mediations	SIB Admin Commands	Use this command to list the mediations on a named SIB bus.	None	• Parameters:
				bus
				name of the SIB bus where the mediations to be listed are to be found (String, required)
				• Returns: A list of SIB mediations.
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask listSIBMediations {-bus bus_name}`
					• Using Jython:
					`AdminTask.listSIBMediations ('[-bus bus_name]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask listSIBMediations {-interactive}`
					• Using Jython:
					`AdminTask .listSIBMediations ('[-interactive]')`

list SI Bus Members	SIB Admin Commands	Use this command to list all servers and clusters which are members of the named SIB bus.	None	• Parameters:
				bus
				name of the SIB bus whose members are to be listed (String, required)
				• Returns: List containing the IDs of bus members – servers and clusters.
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask listSIBusMembers {-bus bus_name}`
					• Using Jython:
					`AdminTask .listSIBusMembers ('[-bus bus_name]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask listSIBusMembers {-interactive}`
					• Using Jython:
					`AdminTask .listSIBusMembers ('[-interactive]')`

list SI Buses	SIB Admin Commands	Use this command to list all SIB buses in the cell.	None	• Parameters: None
				• Returns: None
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask listSIBuses`
					• Using Jython:
					`AdminTask.listSIBuses()`

list SSL Repertoires	None	The **list SSL Repertoires** command lists all of the Secure Sockets Layer (SSL) configuration instances you can associate with an SSL inbound channel.	SSL Inbound Channel instance for which the SSL Config candidates are listed.	• Parameters: None • Returns: A list of eligible SSL configuration object names.	**Batch mode example usage:** • Using Jacl: $AdminTask listSSLRepertoires SSL_3 (*cells/rohitbuildCell01/nodes/rohitbuildNode01/servers/server2|server.xml#SSLInboundChannel_1093445762330*) • Using Jython: AdminTask.listSSLRepertoires('SSL_3(*cells/rohitbuildCell01/nodes/rohitbuildNode01/servers/server2|server.xml#SSLInboundChannel_1093445762330*)') **Interactive mode example usage:** • Using Jacl: $AdminTask listSSLRepertoires {-interactive} • Using Jython: AdminTask.listSSLRepertoires('[-interactive]')
list Server Templates	None	Use the **list Server Templates** command to query available server templates based on server type, platform, or release level.	None	• Parameters: **-serverType** (String, optional) **-platform** (String, optional) **-release Version** (String, optional) • Returns: A list of server template identifications that match with the criteria that you specify with the command parameters. If you do no specify any parameters, all server templates are returned.	**Batch mode example usage:** • Using Jacl: $AdminTask listServerTemplates {-serverType *server_Type*} • Using Jython: AdminTask.listServerTemplates('[-serverType *server_Type*]') **Interactive mode example usage:** • Using Jacl: $AdminTask listServerTemplates {-interactive} • Using Jython: AdminTask.listServerTemplates('[-interactive]')

list Server Types	None	Use the **list Server Types** command to query defined server types on a node.	The identification of a node in the cell, `javax .management.ObjectName`. This target object is optional.	• Returns: A list of server types that you can define on a node. If you do not specify the target object, this command returns all of the server types defined in the entire cell.	**Interactive mode example usage:**
					• Using Jacl:
					`$AdminTask listServerTypes {-interactive}`
					• Using Jython:
					`AdminTask.listServerTypes ('[-interactive]')`

list Servers					**Interactive mode example usage:**
					• Using Jacl:
					`$AdminTask listServers {-interactive}`
					• Using Jython:
					`AdminTask.listServers ('[-interactive]')`

list TAM Settings					**Interactive mode example usage:**
					• Using Jacl:
					`$AdminTask listTAMSettings {-interactive}`
					• Using Jython:
					`AdminTask.listTAMSettings ('[-interactive]')`

list TCP End Points	None	The **list TCP End Points** command lists all named end points that can be associated with a TCP inbound channel.	TCP Inbound Channel instance for which named end points candidates are listed. (ObjectName, required)	• Parameters: **- exclude Distinguished** Shows only non-distinguished named end points. This parameter does not require a value. (Boolean, optional) **- unusedOnly** Shows the named end points not in use by other TCP inbound channel instances. This parameter does not require a value. (Boolean, optional) • Returns: A list of object names for the eligible named end points.	**Batch mode example usage:** • Using Jacl: $AdminTask listTCPEndPoints TCP_1(cells /rohitbuildCell01/ nodes /rohitbuildCellManager01 /servers /dmgr|server.xml# TCPInboundChannel_1) $AdminTask listTCPEndPoints TCP_1(cells /rohitbuildCell01/ nodes /rohitbuildCellManager01 /servers /dmgr|server.xml# TCPInboundChannel_1) {-excludeDistinguished} $AdminTask listTCPEndPoints TCP_1(cells /rohitbuildCell01/ nodes /rohitbuildCellManager01 /servers /dmgr|server.xml #TCPInboundChannel_1) {-excludeDistinguished -unusedOnly} • Using Jython: AdminTask .listTCPEndPoints ('TCP_1 (cells/rohitbuildCell01 /nodes /rohitbuildCellManager01 /servers /dmgr|server.xml #TCPInboundChannel_1)', '[-excludeDistinguished]') AdminTask .listTCPEndPoints ('TCP_1(cells /rohitbuildCell01/ nodes /rohitbuildCellManager01 /servers /dmgr|server.xml# TCPInboundChannel_1)', '[-excludeDistinguished]') AdminTask.listTCPEndPoints ('TCP_1 (cells/rohitbuildCell01 /nodes /rohitbuildCellManager01 /servers/dmgr|server.xml #TCPInboundChannel_1)', '[-excludeDistinguished -unusedOnly]')

list TCP End Points continued					**Interactive mode example usage:**
					• Using Jacl:
					`$AdminTask`
					`listTCPEndPoints`
					`{-interactive}`
					• Using Jython:
					`AdminTask`
					`.listTCPEndPoints`
					`('[-interactive]')`

list TCP Thread Pools	None	The **list TCP Thread Pools** command lists all of the thread pools that can be associated with a TCP inbound channel or TCP outbound channel.	TCP Inbound Channel or TCP Outbound Channel instance for which Thread Pool candidates are listed. (Object Name, required)	• Parameters: None
				• Returns: A list of eligible thread pool object names.
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask`
					`listTCPThreadPools`
					`TCP_1(cells`
					`/rohitbuildCell01/`
					`nodes`
					`/rohitbuildCellManager01`
					`/servers/dmgr|server.xml#`
					`TCPInboundChannel_1)`
					• Using Jython:
					`AdminTask`
					`.listTCPThreadPools`
					`('TCP_1(cells`
					`/rohitbuildCell01/`
					`nodes`
					`/rohitbuildCellManager01`
					`/servers`
					`/dmgr|server.xml#`
					`TCPInboundChannel_1)')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask`
					`listTCPThreadPools`
					`{-interactive}`
					• Using Jython:
					`AdminTask`
					`.listTCPThreadPools`
					`('[-interactive]')`

list Unmanaged Nodes	Unmanaged Node Commands group	Use the **list Unmanaged Nodes** command to list the unmanaged nodes in a configuration.	None	• Parameters: None
				• Returns: List
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask listUnmanagedNodes`
					• Using Jython:
					`AdminTask .listUnmanagedNodes()`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask listUnmanagedNodes {-interactive}`
					• Using Jython:
					`AdminTask .listUnmanagedNodes ('[-interactive]')`

mediate SIB Destination	SIB Admin Commands	Use the **mediate SIB Destination** command to mediate a bus destination. The bus, destination, and mediation definitions must exist prior to using this command. The destination must not be mediated already.	None	• Parameters:
				bus
				the name of the bus where the destination is to be mediated (String, required)
				destinationName
				the name of the destination to be mediated (String, required)
				mediationName
				the name to be given to the mediation (String, required)
				node
				if mediating a destination to a server, specify the node and server name, but not the cluster name (String, optional)
				server
				if mediating a destination to a server, specify the node and server name, but not the cluster name (String, optional)
				cluster
				if mediating a destination to a cluster, specify the cluster name, but not the node or server name (String, optional)
				• Returns: None
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask mediateSIBDestination {-bus busname -name destname -mediationName mediationName}`
					• Using Jython:
					`AdminTask.mediateSIBDestination ('[-bus busname -name destname -mediationName mediationName]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask mediateSIBDestination {-interactive}`
					• Using Jython:
					`AdminTask.mediateSIBDestination ('[-interactive]')`

modify Node Group	Node Group Commands group	The **modify Node Group** command modifies the configuration of a node group. The node group name can not be changed. However, its short name and description are allowed. Also, its node membership can be modified.	The target object is the node group name. This target object is required.	• Parameters:
				- shortName The short name of the node group. This parameter is optional.
				- description The description of the node group. This parameter is optional.
				• Returns: Node group object ID.
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask modifyNodeGroup WBINodeGroup {-shortName WBIGroup -description "Default node group"}`
					• Using Jython:
					`AdminTask.modifyNodeGroup WBINodeGroup('[-shortName WBIGroup -description "WBI" node group]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask modifyNodeGroup {-interactive}`
					• Using Jython:
					`AdminTask.modifyNodeGroup ('[-interactive]')`

modify Node Group Property	Node Group Commands group	The **modify Node Group Property** command modifies custom properties for a node group	The name of the node group. This target object is required.	• Parameters:
				- name The name of the custom property to modify. This parameter is required.
				- value The value of the custom property. This parameter is optional.
				- description The description of the custom property. This parameter is optional.
				• Returns: Properties object ID
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask modifyNodeGroupProperty WBINodeGroup {-name Channel -value "channel1"}`
					• Using Jython:
					`AdminTask.modifyNodeGroupProperty ('WBINodeGroup', '[-name Channel -value channel1]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask modifyNodeGroupProperty {-interactive}`
					• Using Jython:
					`AdminTask.modifyNodeGroupProperty ('[-interactive]')`

modify SIB Destination	SIB Admin Commands	Use the **modify SIB Destination** command to modify the attributes of a SIB destination. The bus and name parameters are used to identify the SIB destination and cannot be modified.	None	• Parameters:
				bus
				bus name (String, required)
				name
				destination name (String, required)
				description
				description (String, optional)
				reliability
				the reliability quality of service for message flows through this destination, from BEST_EFFORT_NONPERSISTENT to ASSURED_PERSISTENT, in order of increasing reliability. Higher levels of reliability have higher impacts on the performance (String, optional)
				maxReliability
				the maximum reliability quality of service that is accepted for values specified by producers (String, optional)
				overrideOf QOSByProducerAllowed
				controls the quality of service for message flows between producers and the destination. Select this option to use the quality of service specified by producers instead of the quality defined for the destination (String, optional)
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask modifySIBDestination {-bus` *`busname`* `-name` *`destname`*`}`
					• Using Jython:
					`AdminTask.modifySIBDestination ('[-bus` *`busname`* `-name` *`destname`*`]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask modifySIBDestination {-interactive}`
					• Using Jython:
					`AdminTask.modifySIBDestination ('[-interactive]')`

modify SIB Destination continued				• Parameters: **defaultPriority** the default priority for message flows through this destination, in the range 0 (lowest) through 9 (highest). This default priority is used for messages that do not contain a priority value (Integer, optional) **maxFailed Deliveries** the maximum number of times that service tries to deliver a message to the destination before forwarding it to the exception destination (Integer, optional) **exception Destination** the name of another destination to which the system sends a message that cannot be delivered to this destination within the specified maximum number of failed deliveries (String, optional) **sendAllowed** clear this option (setting it to false) to stop producers from being able to send messages to this destination (String, optional) **receiveAllowed** clear this option (setting it to false) to prevent consumers from being able to receive messages from	

modify SIB Engine	SIB Admin Commands	Use the **modify SIB Engine** command to modify the attributes of a bus messaging engine. The bus, node, server, cluster and engine parameters are used to identify the engine and cannot be modified. A server can only have one messaging engine, so when using this command to modify a messaging engine from a server there‘s no need to supply the engine name. However, since a cluster can have more than one messaging engine, the engine‘s name must be supplied.	None	- Parameters:
				bus
				the name of the bus to which the messaging engine is to belong (String, required)
				node
				to modify a messaging engine on a server, supply node and server name, but not cluster name (String, optional)
				server
				to modify a messaging engine on a server, supply node and server name, but not cluster name (String, optional)
				cluster
				to modify a messaging engine on a cluster, supply cluster name, but not node and server name (String, optional)
				engine
				the name of the engine to be modified. This is only required if the engine belongs to a cluster (String, optional)
				description
				description of the messaging engine (String, optional)
					Batch mode example usage:
					- Using Jacl:
					`$AdminTask modifySIBEngine {-bus` *`busname`* `-node` *`nodeName`* `-server` *`severname`*`}`
					- Using Jython:
					`AdminTask.modifySIBEngine ('[-bus` *`busname`* `-node` *`nodeName`* `-server` *`severname`*`]')`
					Interactive mode example usage:
					- Using Jacl:
					`$AdminTask modifySIBEngine {-interactive}`
					- Using Jython:
					`AdminTask.modifySIBEngine ('[-interactive]')`

modify SIB Engine continued				• Parameters: **initialState** whether the messaging engine is started or stopped when the associated application server is first started. Until started, the messaging engine is unavailable. (Stopped | Started) (String, optional) **destinationHighMsgs** the maximum total number of messages that the messaging engine can place on its message points (Long, optional) • Returns: None	

modify SIB JMS Activation Spec	SIB JMS Admin Commands	Use the **modify SIB JMS Activation Spec** command to modify the properties of an activation specification.	None	• Parameters:
				name The name of the activation specification that you want to modify. (String, (required)
				propertyList A list of name-value pairs. (required)
				• Returns: None
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask modifySIBJMSActivationSpec {-name specname -propertyList propertyList}`
					• Using Jython:
					`AdminTask.modifySIBJMSActivationSpec ('[-name specname -propertyList propertyList]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask modifySIBJMSActivationSpec {-interactive}`
					• Using Jython:
					`AdminTask.modifySIBJMSActivationSpec ('[-interactive]')`

modify SIB JMS Connection Factory	SIB JMS Admin Commands	Use the **modify SIB JMS Connection Factory** command to modify a unified JMS connection factory at the current scope.	None	- Parameters:
				name
				The name of the SIB JMS connection factory. (String, required)
				jndiName
				The JNDI name of the SIB JMS connection factory. (String, required)
				type
				The type of connection factory to modify. To modify a queue connection factory, set the value to Queue. To modify a topic connection factory, set the value to Topic. If you want to create a generic connection factory, do not specify this option. (String, optional)
				busName
				the SIB bus name (String, optional)
				category
				Classifies or groups the connection factory. (String, optional)
				clientID
				A user-defined string. Only required for durable subscriptions. (String, optional)
				connection Proximity
				The proximity of acceptable messaging engines. Valid values include: Bus, Host, Cluster and Server. (String, optional)
				description
					Batch mode example usage:
					- Using Jacl:
					`$AdminTask modifySIBJMSConnectionFactory {-name *factory_name* -jndiName *jndi_name*}`
					- Using Jython:
					`AdminTask .modifySIBJMSConnectionFactory ('[-name *factory_name* -jndiName *jndi_name*]')`
					Interactive mode example usage:
					- Using Jacl:
					`$AdminTask modifySIBJMSConnectionFactory {-interactive}`
					- Using Jython:
					`AdminTask .modifySIBJMSConnectionFactory ('[-interactive]')`

modify SIB JMS Connection Factory continued				• Parameters: **durable Subscription Home** The durable subscription home value. (String, optional) **nonPersistent Mapping** The non-persistent mapping value. Valid values are BestEffort NonPersistent, Express NonPersistent, Reliable NonPersistent, ReliablePersistent, AssuredPersistent, AsSIBDestination and None. (String, optional) **password** The password that is used to modify connections from the connection factory. (String, optional) **provider EndPoints** A list of endpoint triplets separated by commas. For example: `host:port:chain` (String, optional) **readAhead** The read ahead value. Valid values include: Default, AlwaysOn, and AlwaysOff. (String, optional) **remoteProtocol** The name of the protocol used to connect to a remote messaging engine. (String, optional) **remoteTarget Group** (String, optional) **remoteTargetType** (String, optional)	

modify SIB JMS Queue	SIB JMS Admin Commands	Use the **modify SIB JMS Queue** command to modify a unified JMS queue at the current scope.	None	• Parameters:
				name
				The name of the SIB JMS queue. (String, required)
				jndiName
				The JNDI name of the SIB JMS queue. (String, required)
				description
				A description of the SIB JMS queue (String, optional)
				queueName
				The name of the underlying SIB queue to which the queue points (String, required)
				deliveryMode
				The delivery mode for messages. Legal values are ″Application″, ″NonPersistent″ and ″Persistent″ (String, optional)
				timeToLive
				the time in milliseconds to be used for message expiration (Long, optional)
				priority
				the priority for messages. Whole number in the range 0 to 9 (Integer, optional)
				readAhead
				read-ahead value. Legal values are ″AsConnection″, ″AlwaysOn″ and ″AlwaysOff″ (String, optional)
				timeToLive
				(optional)
				• Returns: None
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask modifySIBJMSQueue {-name` *queue_name* `-jndiName` *jndi_name* `-queueName` *queue_name*`}`
					• Using Jython:
					`AdminTask.modifySIBJMSQueue ('[-name` *queue_name* `-jndiName` *jndi_name* `-queueName` *queue_name*`]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask modifySIBJMSQueue {-interactive}`
					• Using Jython:
					`AdminTask.modifySIBJMSQueue ('[-interactive]')`

modify SIB JMS Topic	SIB JMS Admin Commands	Use the **modify SIB JMS Topic** command to modify the JMS topic at the current scope.	None	• Parameters:
				name
				The name of the SIB JMS topic (String, required)
				jndiName
				the SIB JMS topic's JNDI name (String, required)
				description
				a description of the SIB JMS queue (String, optional)
				topicSpace
				the name of the underlying SIB topic space to which the topic points (String, required)
				***topicName**
				the topic to be used inside the topic space (for example, stock/IBM) (String, required)
				deliveryMode
				the delivery mode for messages. Legal values are ″Application″, ″NonPersistent″ and ″Persistent″ (String, optional)
				timeToLive
				the time in milliseconds to be used for message expiration (Long, optional)
				priority
				the priority for messages. Whole number in the range 0 to 9 (Integer, optional)
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask modifySIBJMSTopic {-name topic_name -jndiName jndi_name -topicName topic_name -topicSpace topicspace_name}`
					• Using Jython:
					`AdminTask.modifySIBJMSTopic ('[-name topic_name -jndiName jndi_name -topicName topic_name -topicSpace topicspace_name]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask modifySIBJMSTopic {-interactive}`
					• Using Jython:
					`AdminTask.modifySIBJMSTopic ('[-interactive]')`

modify SIB JMS Topic continued				- Parameters: **readAhead** read-ahead value. Legal values are ″AsConnection″, ″AlwaysOn″ and ″AlwaysOff″ (String, optional) **busName** the name of the bus on which the topic resides (String, optional) - Returns: None	

modify SIB Mediation	SIB Admin Commands	Use this command to modify the attributes of a SIB mediation. The bus and mediation Name parameters identify the mediation and cannot be modified.	None	- Parameters:
				bus the name of the bus that owns the mediation (String, required)
				mediationName name of the mediation to be modified (String, required)
				description description of the mediation (String, optional)
				handlerList Name the name of the handler list that was defined when the mediation was deployed (String, optional)
				globalTransaction whether or not a global transaction is started for each message processed (Boolean, optional)
				allowConcurrent Mediation whether or not to apply the mediation to multiple messages concurrently, and preserve message ordering (Boolean, optional)
				selector the text string that must be present in a message for the mediation to process the message (String, optional)
				discriminator the text string that must not be present in a message for the mediation to process the message (String
					Batch mode example usage:
					- Using Jacl:
					`$AdminTask modifySIBMediation {-bus` *`bus_name`* `-jndiName` *`jndi_name`*`}`
					- Using Jython:
					`AdminTask.modifySIBMediation ('[-bus` *`bus_name`* `-mediationName` *`mediation_name`*`]')`
					Interactive mode example usage:
					- Using Jacl:
					`$AdminTask modifySIBMediation {-interactive}`
					- Using Jython:
					`AdminTask.modifySIBMediation ('[-interactive]')`

modify SIBus	SIB Admin Commands	Use this command to modify the attributes of the named bus. He “bus” parameter identifies the bus to be modified, and is not used to change the name of the bus.	None	- Parameters:
				bus
				name of bus to modify (String, required)
				description
				description of bus modify (String, optional)
				secure
				enable or disable bus security (Boolean, optional)
				interEngineAuthAlias
				name of the authentication alias used to authorize communication between messaging engines on the bus.
				mediations AuthAlias
				name of the authentication alias used to authorize mediations to access the bus (String, optional)
				protocol
				the protocol used to send and receive messages between messaging engines, and between API clients and messaging engines (String, optional)
					Batch mode example usage:
					- Using Jacl:
					`$AdminTask modifySIBus {-bus busname -description text -secure True -mediationsAuthAlias name -protocol protocol -discardOnDelete False}`
					- Using Jython:
					`AdminTask.modifySIBus ('[-busbusname -description "text" -secure True -mediationsAuthAlias name -protocol protocol -discardOnDelete False]')`
					Interactive mode example usage:
					- Using Jacl:
					`$AdminTask modifySIBus {-interactive}`
					- Using Jython:
					`AdminTask.modifySIBus ('[-interactive]')`

modify SIBus continued				• Parameters: **discardOnDelete** indicate whether or not any messages left in a queue’s data store should be discarded when the queue is deleted (Boolean, optional) **destinationHighMsgs** the maximum number of messages that any queue on the bus can hold (Long, optional) **configurationReloadEnabled** indicate whether configuration files should be dynamically reloaded for this bus (Boolean, optional) • Returns: None	

modify SIBus Member	SIB Admin Commands	Use this command to modify the attributes of the bus member identified by the bus, node, server and cluster parameters.	None	- Parameters:
				bus
				name of bus to which the member belongs(String, required)
				node
				to specify a server bus member, supply node and server name, but not cluster name (String, optional)
				server
				to specify a server bus member, supply node and server name, but not cluster name (String, optional)
				cluster
				to specify a cluster bus member, supply cluster name but not node and server name (String, optional)
				- Returns: None
					Batch mode example usage:
					- Using Jacl:
					`$AdminTask modifySIBusMember {-bus `*`busname`*` -node `*`nodename`*` -server `*`servername`*` -description `*`text`*`}`
					- Using Jython:
					`AdminTask.modifySIBusMember ('[-bus `*`busname`*` -node `*`nodename`*` -server `*`servername`*` -description "`*`text`*`"]')`
					Interactive mode example usage:
					- Using Jacl:
					`$AdminTask modifySIBusMember {-interactive}`
					- Using Jython:
					`AdminTask.modifySIBusMember ('[-interactive]')`

move Cluster To Core Group	Core Group Management group	The **move Cluster To Core Group** command moves all of servers in a cluster that you specify from a core group to another core group. All of the servers in cluster must be members of the same core group.	None	• Parameters:
				- source The name of the core group that contains the cluster that you want to move. The core group must exist prior to running this command. The cluster that you specify must be a member of this core group. (String, required)
				- target The name of the core group where you want to move the cluster. (String, required)
				- clusterName The name of the cluster that you want to move. (String, required)
				• Returns: None
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask moveClusterToCoreGroup {-source OldCoreGroup -target NewCoreGroup -clusterName ClusterOne}`
					• Using Jython:
					`AdminTask.moveClusterToCoreGroup ('[-source OldCoreGroup -target NewCoreGroup -clusterName ClusterOne]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask moveClusterToCoreGroup {-interactive}`
					• Using Jython:
					`AdminTask.moveClusterToCoreGroup ('[-interactive]')`

move Server To Core Group	Core Group Management group	The **move Server To Core Group** command moves a server to a core group that you specify. When the server is added to the core group that you specify, it will be removed from the core group where it originally resided.	None	• Parameters: **- source** The name of the core group that contains the server that you want to move. The core group must already exist with the server that you specify being a member of the core group. (String, required) **- target** The name of the core group where you want to move the server. The core group that you specify must exist prior to running the command. (String, required) **- nodeName** The name of the node that contains the server that you want to move. (String, required) **- serverName** The name of the server that you want to move. (String, required) • Returns: None	**Batch mode example usage:** • Using Jacl: $AdminTask moveServerToCoreGroup {-source *OldCoreGroup* -target *NewCoreGroup* -nodeName *myNode* -serverName *myServer*} • Using Jython: AdminTask.moveServerToCoreGroup ('[-source *OldCoreGroup* -target *NewCoreGroup* -nodeName *myNode* -serverName *myServer*]') **Interactive mode example usage:** • Using Jacl: $AdminTask moveServerToCoreGroup {-interactive} • Using Jython: AdminTask.moveServerToCoreGroup ('[-interactive]')

publish SIB WS Inbound Service	SIB Web Services group	The publish SIB WS Inbound Service command publishes the WSDL document for the inbound service and the associated ports to the registry and business defined by the UDDI Publication object.	The object name of the inbound service object.	• Parameters:
				uddiPublication
				The name of the UDDI publication for the service. (required)
				userId
				The user ID to use to retrieve the WSDL. (optional)
				password
				The password to use to retrieve the WSDL. (optional)
				• Returns: None
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask publishSIBWSInboundService $inService {-uddiPublication "MyUddi"}`
					• Using Jython:
					`AdminTask.publishSIBWSInboundService (inService, '[-uddiPublication MyUddi]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask publishSIBWSInboundService {-interactive}`
					• Using Jython:
					`AdminTask.publishSIBWSInboundService ('[-interactive]')`

reconfigure TAM					**Interactive mode example usage:**
					• Using Jacl:
					`$AdminTask reconfigureTAM {-interactive}`
					• Using Jython:
					`AdminTask.reconfigureTAM ('[-interactive]')`

refresh SIB WS Inbound Service WSDL	SIB Web Services group	The **refresh SIB WS Inbound Service WSDL** command loads the WSDL document from the WSDL Location parameters of the inbound service and locates the WSDL Location-specified service element. If the service element is not present, this command fails. If the outbound ports are not a subset of the ports in the loaded WSDL document, this command fails.
		If the WSDL will be retrieved through a proxy, the server on which the command is running must have the system properties that identify the proxy server set correctly.
			The object name of the inbound service object.	• Parameters:
				userId
				The user ID to use to retrieve the WSDL. (optional)
				password
				The password to use to retrieve the WSDL. (optional)
				• Returns: None
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask refreshSIBWSInboundServiceWSDL $inService`
					• Using Jython:
					`AdminTask .refreshSIBWSInboundServiceWSDL (inService)`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask refreshSIBWSInboundServiceWSDL {-interactive}`
					• Using Jython:
					`AdminTask .refreshSIBWSInboundServiceWSDL ('[-interactive]')`

refresh SIB WS Outbound Service WSDL	SIB Web Services group	The **refresh SIB WS Outbound Service WSDL** command loads the WSDL document from the WSDL Location parameters of the outbound service and locates the WSDL Location specified service element. If the service element is not present, this command fails. If the outbound ports are not a subset of the ports in the loaded WSDL document, this command fails.
		If the WSDL will be retrieved through a proxy, the server on which the command is running must have the system properties that identify the proxy server set correctly.
			The object name of the outbound service object.	• Parameters:
				userId The user ID to use to retrieve the WSDL. (optional)
				password The password to use to retrieve the WSDL. (optional)
				• Returns: None
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask refreshSIBWSOutboundServiceWSDL $outService`
					• Using Jython:
					`AdminTask .refreshSIBWSOutboundServiceWSDL (outService)`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask refreshSIBWSOutboundServiceWSDL {-interactive}`
					• Using Jython:
					`AdminTask .refreshSIBWSOutboundServiceWSDL ('[-interactive]')`

remove Node Group	Node Group Commands group	The **remove Node Group** command removes the configuration of a node group. You can remove a node group if it does not contain any members. Also, the default node group can not be removed.	The name of the node group to be removed. This target object is required.	• Parameters: None
				• Returns: Node group object ID.
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask removeNodeGroup WBINodeGroup`
					• Using Jython:
					`AdminTask.removeNodeGroup ('WBINodeGroup')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask removeNodeGroup {-interactive}`
					• Using Jython:
					`AdminTask.removeNodeGroup ('[-interactive]')`

remove Node Group Member	Node Group Commands group	The **remove Node Group Member** command removes the configuration of a node group member.
		• A node must always be a member of at least one node group.
		• You cannot remove a node from a node group that is part of a cluster in that node group.
			The target object is the node group containing the member to be removed. This target object is required.	• Parameters:
				- nodeName The name of the node to be removed from a node group. This parameter is required.
				• Returns: Node group member object ID.
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask removeNodeGroupMember WBINodeGroup {-nodeName WBINode}`
					• Using Jython:
					`AdminTask.removeNodeGroupMember ('WBINodeGroup', '[-nodeName WBINode]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask removeNodeGroupMember {-interactive}`
					• Using Jython:
					`AdminTask.removeNodeGroupMember ('[-interactive]')`

remove Node Group Property	Node Group Commands group	The **remove Node Group Property** command removes custom properties of a node group.	The name of the node group. This target object is required.	• Parameters:
				- name The name of the custom property to remove. This parameter is required.
				• Returns: Properties object ID
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask removeNodeGroupProperty WBINodeGroup {-name Channel}`
					• Using Jython:
					`AdminTask.removeNodeGroupProperty ('WBINodeGroup', '[-name Channel]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask removeNodeGroupProperty {-interactive}`
					• Using Jython:
					`AdminTask.removeNodeGroupProperty ('[-interactive]')`

remove SIB WS Inbound Port	SIB Web Services group	The **remove SIB WS Inbound Port** command removes the configuration of an inbound port.	The object name of the inbound port object that you want to remove.	• Parameters: None
				• Returns: None
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask removeSIBWSInboundPort $inPort`
					• Using Jython:
					`AdminTask .removeSIBWSInboundPort(inPort)`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask removeSIBWSInboundPort {-interactive}`
					• Using Jython:
					`AdminTask.removeSIBWSInboundPort ('[-interactive]')`

remove SIB WS Outbound Port	SIB Web Services group	The remove SIB WS Outbound Port command removes the configuration of an outbound port. If the port that you delete is the default port for the outbound service, one of the remaining ports, if any, will be chosen as the new default. Resources that are associated with the outbound port, for example, WS-Security configuration, are disassociated from the outbound port but not deleted.	Object name of the outbound port object that you want to remove.	• Parameters: None
				• Returns: None
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask removeSIBWSOutboundPort`
					`$outPort`
					• Using Jython:
					`AdminTask.removeSIBWSOutboundPort`
					`(outPort)`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask removeSIBWSOutboundPort`
					`{-interactive}`
					• Using Jython:
					`AdminTask.removeSIBWSOutboundPort`
					`('[-interactive]')`

remove SI Bus Member	SIB Admin Commands	Use this command to remove a server or a cluster from a SIB bus. As well as removing the server or cluster from the SIB bus, this command also deletes all SIB messaging engines associated with the bus, all queue points and publication points owned by those engines, and all queue point references and publication point references which refer to the deleted queue points and publication points.	None	- Parameters: **bus** name of SIB bus to remove member from (String, required) **node** to specify a server bus member, supply node and server name, but not cluster name (String, optional) **server** to specify a server bus member, supply node and server name, but not cluster name (String, optional) **cluster** to specify a cluster bus member, supply cluster name but not node and server name (String, optional) - Returns:	**Batch mode example usage:** - Using Jacl: $AdminTask removeSIBusMember {-bus *busname* -node *nodename* -server *servername*} - Using Jython: AdminTask.removeSIBusMember ('[-bus *busname* -node *nodename* -server *servername*]') **Interactive mode example usage:** - Using Jacl: $AdminTask removeSIBusMember {-interactive} - Using Jython: AdminTask.removeSIBusMember ('[-interactive]')
remove Unmanaged Node	Unmanaged Node Commands group	Use the **remove Unmanaged Node** command to remove an unmanaged node from the configuation.	None	- Parameters: **- nodeName** The name of the unmanaged node. (String, required) - Returns: null	**Batch mode example usage:** - Using Jacl: $AdminTask removeUnmanagedNode {-nodeName myNode } - Using Jython: AdminTask.removeUnmanagedNode ('[-nodeName myNode]') **Interactive mode example usage:** - Using Jacl: $AdminTask removeUnmanagedNode {-interactive} - Using Jython: AdminTask.createUnmanagedNode ('[-interactive]')

remove WS GW Target Service	WS Gateway group	The **remove WS GW Target Service** command removes a target service from the gateway service. The destinations that are associated with the target service are not deleted. If the target service that you remove is the default target service, the default is set to the first target service in the set or cleared if there are none left.	object name of the Target Service object	• Parameters: None
				• Returns: None
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask removeWSGWTargetService $gwTarget`
					• Using Jython:
					`AdminTask.removeWSGWTargetService (gwTarget)`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask removeWSGWTargetService {-interactive}`
					• Using Jython:
					`AdminTask.removeWSGWTargetService ('[-interactive]')`

set Default SIB WS Outbound Port	SIB Web Services group	The **set Default SIB WS Outbound Port** command updates the default outbound port for an outbound service.	The object name of the outbound service whose default port you want to update.	• Parameters:
				name
				The name of the port that you want to set as the default. (required)
				• Returns: None
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask setDefaultSIBWSOutboundPort $outService {-name "MyServiceSoap"}`
					• Using Jython:
					`AdminTask .setDefaultSIBWSOutboundPort (outService, '[-name MyServiceSoap]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask setDefaultSIBWSOutboundPort {-interactive}`
					• Using Jython:
					`AdminTask .setDefaultSIBWSOutboundPort ('[-interactive]')`

show SIB Destination	SIB Admin Commands	Use the **show SIB Destination** command to get the attribute names/values of a SIB destination. The bus and name parameter identify the SIB destination whose attributes are required.	None	• Parameters:
				bus
				bus name (String, required)
				name
				destination name (String, required)
				• Returns: The attribute names and values of the named SIB destination on the named bus.
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask showSIBDestination {-bus busname -name destname}`
					• Using Jython:
					`AdminTask.showSIBDestination ('[-bus busname -name destname]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask showSIBDestination {-interactive}`
					• Using Jython:
					`AdminTask.showSIBDestination ('[-interactive]')`

show SIB Engine	SIB Admin Commands	Use the **show SIB Engine** command to get the attribute names/values of a SIB messaging engine belonging to a given bus member. If the bus member is a server, only the bus, node and server parameters need be supplied. A server only has 1 engine, so the engine parameter is not necessary. If the bus member is a cluster, the bus, cluster and engine parameters must be supplied, since a cluster can have more than one engine.	None	• Parameters:
				bus
				the name of the bus to which the messaging engine to be shown belongs (String, required)
				node
				to show a messaging engine that belongs to a server, supply node and server name, but not cluster name (String, optional)
				server
				to show a messaging engine that belongs to a server, supply node and server name, but not cluster name (String, optional)
				cluster
				to show a messaging engine that belongs to a cluster, supply cluster name, but not node and server name (String, optional)
				engine
				The name of the engine to show. If the bus member has only one messaging engine, you do not need to specify the engine option. If the bus member has several messaging engines, you must specify the name of the engine for which you want to display details. (String, optional)
				• Returns: The attribute names and values of the identified SIB messaging engine.
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask showSIBEngine {-bus busname -node nodeName -server severname}`
					• Using Jython:
					`AdminTask.showSIBEngine('[-bus busname -node nodeName -server severname]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask showSIBEngine {-interactive}`
					• Using Jython:
					`AdminTask.showSIBEngine ('[-interactive]')`

show SIB JMS Activation Spec	SIB Admin Commands	The show SIB JMS Activation Spec command shows details about a JMS activation specification.	None	• Parameters:
				bus
				The name of the bus that owns the mediation (String, required)
				mediationName
				The name of the mediation to be shown (String, required)
				• Returns: A list
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask showSIBJMSActivationSpec {-bus bus_name -mediationName mediation_name}`
					• Using Jython:
					`AdminTask.showSIBJMSActivationSpec ('[-bus bus_name -mediationName mediation_name]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask showSIBJMSActivationSpec {-interactive}`
					• Using Jython:
					`AdminTask.showSIBJMSActivationSpec ('[-interactive]')`

show SIB JMS Connection Factory	SIB JMS Admin Commands	The **show SIB JMS Connection Factory** command shows details about a JMS connection factory.	None	• Parameters:
				name
				The name of the SIB JMS connection factory (String, required)
				• Returns: A set of property value pairs for the JMS connection factory that you specified.
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask showSIBJMSConnectionFactory {-name factory_name}`
					• Using Jython:
					`AdminTask .showSIBJMSConnectionFactory ('[-name factory_name]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask showSIBJMSConnectionFactory {-interactive}`
					• Using Jython:
					`AdminTask .showSIBJMSConnectionFactory ('[-interactive]')`

show SIB JMS Queue	SIB JMS Admin Commands	Use the **show SIB JMS Queue** command to show the details about a JMS queue.	None	- Parameters: **name** The name of the SIB JMS queue. (String, required) - Returns: A set of property value pairs for the JMS queue that you specified.	**Batch mode example usage:** - Using Jacl: $AdminTask showSIBJMSQueue {-name *queue_name*} - Using Jython: AdminTask.showSIBJMSQueue ('[-name *queue_name*]') **Interactive mode example usage:** - Using Jacl: $AdminTask showSIBJMSQueue {-interactive} - Using Jython: AdminTask.showSIBJMSQueue ('[-interactive]')
show SIB JMS Topic	SIB JMS Admin Commands	Use this command to show the details for a JMS topic.	None	- Parameters: **- name** The name of the SIB JMS topic (String, required) - Returns: A set of property value pairs for the JMS topic that you specified.	**Batch mode example usage:** - Using Jacl: $AdminTask showSIBJMSTopic {-name *topic_name*} - Using Jython: AdminTask.showSIBJMSTopic ('[-name *topic_name*]') **Interactive mode example usage:** - Using Jacl: $AdminTask showSIBJMSTopic {-interactive} - Using Jython: AdminTask.showSIBJMSTopic ('[-interactive]')
show SIB Mediation	SIB Admin Commands	Use this command to get the attribute names/values of a SIB mediation.	None	- Parameters: **bus** the name of the bus that owns the mediation (String, required) **mediationName** the name of the mediation to be shown (String, required) - Returns: The attribute names and values of the identified SIB mediation.	**Batch mode example usage:** - Using Jacl: $AdminTask showSIBMediation {-bus *bus_name* -mediationName *mediation_name*} - Using Jython: AdminTask.showSIBMediation ('[-bus *bus_name* -mediationName *mediation_name*]') **Interactive mode example usage:** - Using Jacl: $AdminTask showSIBMediation {-interactive} - Using Jython: AdminTask.showSIBMediation ('[-interactive]')

show SIBus	SIB Admin Commands	Use this command to get the attribute names/values of a SIB bus.	None	• Parameters:
				bus
				bus name (String, required)
				• Returns: The attribute names and values of the identified SIB bus.
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask showSIBus {-bus bus_name}`
					• Using Jython:
					`AdminTask.showSIBus('[-bus bus_name]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask showSIBus {-interactive}`
					• Using Jython:
					`AdminTask.showSIBus ('[-interactive]')`

show SIBus Member	SIB Admin Commands	Use this command to get the attribute names/values of a SIB bus member.	None	• Parameters:
				bus
				name of bus to show member from (String, required)
				node
				to specify a server bus member, supply node and server name, but not cluster name (String, optional)
				server
				to specify a server bus member, supply node and server name, but not cluster name (String, optional)
				cluster
				to specify a cluster bus member, supply cluster name but not node and server name (String, optional)
				• Returns: The attribute names and values of the identified SIB bus member.
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask showSIBusMember {-bus busname -node nodename -server servername}`
					• Using Jython:
					`AdminTask.showSIBusMember ('[-bus busname -node nodename -server servername]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask showSIBusMember {-interactive}`
					• Using Jython:
					`AdminTask.showSIBusMember ('[-interactive]')`

show Server Info					**Interactive mode example usage:**
					• Using Jacl:
					`$AdminTask showServerInfo {-interactive}`
					• Using Jython:
					`AdminTask.showServerInfo ('[-interactive]')`

show Server Type Info					**Interactive mode example usage:**
					• Using Jacl:
					`$AdminTask showServerTypeInfo {-interactive}`
					• Using Jython:
					`AdminTask.showServerTypeInfo ('[-interactive]')`

show Template Info	None	Use the **show Template Info** command to query metadata information for a specific template. This command will only work for server templates.	The identification of a server template, `javax.management.ObjectName`. This target object is required.	• Returns: A property object that holds the metadata information regarding a specific template.	**Interactive mode example usage:**
					• Using Jacl:
					`$AdminTask showTemplateInfo {-interactive}`
					• Using Jython:
					`AdminTask.showTemplateInfo ('[-interactive]')`

unconfigure TAM					**Interactive mode example usage:**
					• Using Jacl:
					`$AdminTask unconfigureTAM {-interactive}`
					• Using Jython:
					`AdminTask.unconfigureTAM ('[-interactive]')`

unmediate SIB Destination	SIB Admin Commands	Use this command to unmediate the named destination on the named bus. Unmediating a destination simply removes the association between a SIB destination and a SIB mediation.	None	• Parameters:
				bus
				the name of the bus where the destination is currently mediated (String, required)
				destinationName
				the name of the destination to be unmediated (String, required)
				• Returns: None
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask unmediateSIBDestination {-bus bus_name -destinationName destination_name}`
					• Using Jython:
					`AdminTask.unmediateSIBDestination ('[-bus bus_name -destinationName destination_name]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask unmediateSIBDestination {-interactive}`
					• Using Jython:
					`AdminTask.unmediateSIBDestination ('[-interactive]')`

unpublish SIB WS Inbound Service	SIB Web Services group	The **unpublish SIB WS Inbound Service** command removes the WSDL document for the inbound service, including the ports, from the registry and business defined by the UDDI publication object.	The object name of the inbound service object.	• Parameters:
				uddi Publication
				The name of the UDDI publication for the service. (required)
				userId
				The user ID to use to retrieve the WSDL. (optional)
				password
				The password to use to retrieve the WSDL. (optional)
				• Returns: None
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask unpublishSIBWSInboundService $inService {-uddiPublication "MyUddi"}`
					• Using Jython:
					`AdminTask .unpublishSIBWSInboundService (inService, '[-uddiPublication MyUddi]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask unpublishSIBWSInboundService {-interactive}`
					• Using Jython:
					`AdminTask .unpublishSIBWSInboundService ('[-interactive]')`

update App On Cluster	None	The **update App On Cluster** command can be used to synchronize nodes and restart cluster members for an application update deployed to a cluster. After application update, this command can be used to synchronize the nodes without stopping all the cluster members on all the nodes at one time.
		This command synchronizes one node at a time. Each node is synchronized by first stopping the cluster members on which the application is targetted and then performing nodesync operation and then restarting the cluster members.
			None	• Parameters:
				-Application Names The names of the applications that are updated.
				-timeout The timeout value in seconds for each node synchronization. The default is 300 seconds.
				• Returns: None
					Batch mode example usage:
					• Using Jacl:
					`$AdminTask updateAppOnCluster {-ApplicationNames app1}`
					`$AdminTask updateAppOnCluster { -ApplicationNames app1 -timeout 600}`
					• Using Jython:
					`AdminTask.updateAppOnCluster ('[-ApplicationNames app1]')`
					`AdminTask.updateAppOnCluster ('[-ApplicationNames app1 -timeout 600]')`
					Interactive mode example usage:
					• Using Jacl:
					`$AdminTask updateAppOnCluster -interactive`
					• Using Jython:
					`AdminTask.updateAppOnCluster ('[-interactive]')`

update App On Cluster		This command may take more than the default connector timeout period depending on the number of nodes that the target cluster spans. Be sure to set proper timeout values in the soap.client .props file, when a SOAP connector is used, and in the sas.client .props file, when a RMI connector is used. This command is not supported in local mode.			

Administrative command invocation syntax

You can use an administrative command in batch mode or interactive mode. The following is the syntax for using an administrative command:

Using Jacl:

```
$AdminTask cmdName [targetObject] [options]
```

where `options` include:

```
{
  [-paramName paramValue] [-paramName] ...
  [-stepName {{stepParamValue ...} ...} ...]
  [-delete {-stepName {{stepKeyParamValue ...} ...} ...} ...]
  [-interactive]
}
```

Using Jython:

```
AdminTask.cmdName(['targetObject'], [options])
```

where `options` include:

```
'[
[-paramName paramValue] [-paramName ...]
[-stepName [[stepParamValue ...] ...] ...]
[-delete [-collectionStepName [[stepKeyParamValue ...] ...] ...] ...]
[-interactive]
]'
```

where:

cmdName	represents the name of an administrative command to be run.
targetObject	represents the target object on which the command operates. Depending on the administrative command, this input can be required, optional, or nonexistent. This input corresponds to the Target object that is displayed in the command-specific help.
paramName	represents the parameter name of the executed command. Depending on the administrative command, this input can be required, optional, or nonexistent. Each parameter name corresponds to an argument name that is displayed in the Arguments area of the command specific help.
paramValue	represents the parameter value to set for the preceding parameter name. Parameters are specified as name-value pairs. The parameter value is not required if a parameter has Boolean as its value type. If you specify the parameter name only, without specifying a value for a Boolean type parameter, the value is set to true.
stepName	represents the step name of the command. This input corresponds to a step name that is displayed in the Steps area of the command-specific help.
stepParamValue ...	represents the values of the parameters for a step. Provide all the parameter values of a step in the correct order as displayed in the step-specific help. For any optional parameters that you do not want to specify a value, put ″″ instead of the value. If a command step is a collection type, for example, it contains multiple objects where each object has the same set of parameters, then you can specify multiple objects with each object enclosed by a pair of braces. For collection type steps, each step parameter is a key or non-key. Key parameters in a step are used to uniquely identify an object in the collection. If data exists in the step, key parameter values that are provided in the input are compared with key parameter values in the existing data. If a match is found, the existing data is updated. Otherwise, if the specified step allows the addition of new objects, the input values are added.
delete	represents the option to delete existing data from specified step that supports collection.
collectionStepName	represents the collection step name .
stepKeyParamValue ...	represents the values of key parameters to uniquely identify an object to be deleted from a collection step. You must provide the key parameter values of an object in the order that they are displayed in the step specific help.
interactive	represents the option to enter interactive mode.
[ ]	represents a Jython list bracket.
[ ]	indicates that the parameters or options inside the brackets are optional. Do not type these brackets as part of the syntax.

Properties used by scripted administration

This article explains the Java properties that are used by scripted administration. Three levels of default property files load before any property file that is specified on the command line. The first level represents an installation default, located in the properties directory for each WebSphere Application Server profile called wsadmin.properties. The second level represents a user default, and is located in the Java user.home property. This properties file is also called wsadmin.properties. The third level is a properties file that is pointed to by the WSADMIN_PROPERTIES environment variable. This environment variable is defined in the environment where the wsadmin tool starts. If one or more of these property files is present, they are interpreted before any properties file that is present on the command line. These three levels of property files load in the order that they are specified. The properties file that is loaded last overrides the ones loaded earlier.

The following Java properties are used by scripting:

com.ibm.ws.scripting.classpath:

Searches for classes and resources, and is appended to the list of paths.

com.ibm.ws.scripting.connectionType:

Determines the connector to use. This value can either be `SOAP`, `RMI`, or `NONE`. The `wsadmin.properties` file specifies `SOAP` as the connector.

com.ibm.ws.scripting.host:

Determines the host to use when attempting a connection. If not specified, the default is the local machine.

com.ibm.ws.scripting.port:

Specifies the port to use when attempting a connection. The `wsadmin.properties` file specifies `8879` as the SOAP port for a single server installation.

com.ibm.ws.scripting.defaultLang:

Indicates the language to use when running scripts. The `wsadmin.properties` file specifies Jacl as the scripting language.

The supported scripting languages are Jacl and Jython.

com.ibm.ws.scripting.traceString:

Turns on tracing for the scripting process. The default has tracing turned off.

com.ibm.ws.scripting.traceFile:

Determines where trace and log output is directed. The `wsadmin.properties` file specifies the `wsadmin.traceout` file that is located in the `logs` directory of each WebSphere Application Server profile as the value of this property.

If multiple users work with the wsadmin tool simultaneously, set different traceFile properties in the user properties files. If the file name contains double-byte character set (DBCS) characters, use a unicode format, such as `\uxxxx`, where `xxxx` is a number.

com.ibm.ws.scripting.validationOutput:

Determines where the validation reports are directed. The default file is `wsadmin.valout` which is located in the `logs` directory of each WebSphere Application Server profile.

If multiple users work with the wsadmin tool simultaneously, set different validationOutput properties in the user properties files. If the file name contains double-byte character set (DBCS) characters, use unicode format, such as `\uxxxx`, where `xxxx` is a number.

com.ibm.ws.scripting.emitWarningForCustomSecurityPolicy:

Controls whether the WASX7207W message is emitted when custom permissions are found.

The possible values are `true` and `false`. The default value is `true`.

com.ibm.ws.scripting.tempdir:

Determines the directory to use for temporary files when installing applications.

The Java virtual machine API uses `java.io.temp` as the default value.

com.ibm.ws.scripting.validationLevel:

Determines the level of validation to use when configuration changes are made from the scripting interface.

Possible values are: `NONE`, `LOW`, `MEDIUM`, `HIGH`, `HIGHEST`. The default is `HIGHEST`.

com.ibm.ws.scripting.crossDocumentValidationEnabled:

Determines whether the validation mechanism examines other documents when changes are made to one document.

Possible values are `true` and `false`. The default value is `true`.

com.ibm.ws.scripting.profiles:

Specifies a list of profile scripts to run automatically before running user commands, scripts, or an interactive shell.

The `wsadmin.properties` file specifies `securityProcs.jacl` and `LTPA_LDAPSecurityProcs.jacl` as the values of this property. If Jython is specified with the wsadmin -lang option, the wsadmin tool performs a conversion to change the profile script names that are specified in this property to use the file extension that matches the language specified. Use the provided script procedures with the default settings to make security configuration easier.

Using Ant to automate tasks

To support using Apache Ant with Java 2 Platform, Enterprise Edition (J2EE) applications running on the application server, the product provides a copy of the Ant tool and a set of Ant tasks that extend the capabilities of Ant to include product-specific functions. Ant has become a very popular tool among Java programmers.

Apache Ant is a Java-based build tool. In theory, it is similar to Make, but Ant is different. Instead of a model in which it is extended with shell-based commands, Ant is extended using Java classes. Instead of writing shell commands, XML-based configuration files are used. These files reference a target tree in which various tasks are run. Each task is run by an object that implements a particular Task interface.

By combining the following tasks with those provided by Ant, you can create build scripts that compile, package, install, and test your application on the application server:

- Install and uninstall applications
- Start and stop servers in a base configuration
- Run administrative scripts or commands
- Run the Enterprise JavaBeans (EJB) deployment tool
- Run the JavaServer Pages (JSP) file precompilation tool

For more detailed information about Ant, refer to the Apache organization Web site.

- To run Ant and have it automatically see the WebSphere classes, use the “ws_ant command.”
- Use “Ant tasks for deployment and server operation.”

  This topic describes where to find the API documentation for the Apache Ant tasks for deploying applications and operating application servers.
- Use “Ant tasks for building application code.”

  This topic describes where to find the API documentation for the Apache Ant tasks for building applications.

ws_ant command

This topic describes where to find information about the ws_ant command, which is provided for using with Apache Ant, a Java-based build tool that is popular among Java programmers.

In theory, Ant is similar to Make, but Ant is different. Instead of a model in which it is extended with shell-based commands, Ant is extended using Java classes. Instead of writing shell commands, XML-based configuration files are used. These files reference a target tree in which various tasks are run. Each task is run by an object that implements a particular Task interface.

For the Apache Ant tool that is provided by this product, see the following file:

```
install_root/bin/ws_ant.bat|sh
```

Ant tasks for deployment and server operation

This topic describes where to find the API documentation for the Apache Ant tasks for deploying applications and operating application servers.

The Apache Ant tasks for the product reside in the Java package: `com.ibm.websphere.ant.tasks`. The API documentation for this package contains detailed information about all of the Ant tasks that are provided and how to use them. The API documentation is available in the **Reference** section of the information center.

Ant tasks for building application code

This topic describes where to find the documentation for the Ant tasks provided for building application code using the Application Server Toolkit (which is a CD included with WebSphere Application Server as a separately installable toolkit).

Within the Application Server Toolkit product documentation, open the section **Working with Ant**. You can locate the topic by searching for **Working with Ant**, or from the navigation view, select **Help > Help Contents > Developing Java Applications > Developing enterprise applications > J2EE applications > Working with Ant**.

Using administrative programs (JMX)

This topic describes how to use Java application programming interfaces (APIs) to administer WebSphere Application Server and to manage your applications.

You can administer WebSphere Application Server and your applications through tools that come with the product or through programming with the Java APIs.

The wsadmin scripting tool, the administrative console, and the administrative command-line tools come with the product. These administrative tools provide most of the functions that you need to manage the product and the applications that run in WebSphere Application Server. You can use the command-line tools from automation scripts to control the servers. Scripts that are written for the wsadmin scripting tool offer a wide range of possible custom solutions that you can develop quickly.

Investigate these tools with the Java APIs to determine the best ways to administer WebSphere Application Server and your applications. For information on the Java APIs, view Java Management Extensions (JMX) API documentation

WebSphere Application Server supports access to the administrative functions through a set of Java classes and methods. 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.

You can prepare, install, uninstall, edit, and update applications through programming. Preparing an application for installation involves collecting various types of WebSphere Application Server-specific binding information to resolve references that are defined in the application deployment descriptors. This information can also be modified after installation by editing a deployed application. Updating consists of adding, removing or replacing a single file or a single module in an installed application, or supplying a partial application that manipulates an arbitrary set of files and modules in the deployed application. Updating the entire application uninstalls the old application and installs the new one. Uninstalling an application removes it entirely from the WebSphere Application Server configuration.

Perform any or all of the following tasks to manage WebSphere Application Server and your Java 2 Platform, Enterprise Edition (J2EE) applications through programming.

- Create a custom Java administrative client program using the Java administrative APIs.

  This topic describes how to develop a Java program that uses the WebSphere Application Server administrative APIs to access the administrative system of WebSphere Application Server.
- Extend the WebSphere Application Server administrative system with custom MBeans.

  This topic describes how to extend the WebSphere Application Server administration system by supplying and registering new JMX MBeans in one of the Application Server processes. In this case, you can use the administrative classes and methods to add newly managed objects to the administrative system.
- Deploy and manage a custom Java administrative client program for use with multiple Java 2 Platform, Enterprise Edition application servers.

  This topic describes how to connect to a J2EE server, and how to manage multiple vendor servers.
- Manage applications through programming

  This topic describes how, through Java MBean programming, to install, update, and delete a J2EE application on WebSphere Application Server.

Depending on which tasks you complete, you have created your own administrative program, extended the WebSphere Application Server administrative console, connected and managed vendor servers, or managed your applications through programming.

You can continue to administer WebSphere Application Server and your applications through programming or in combination with the tools that come with the WebSphere Application Server.

Creating a custom Java administrative client program using WebSphere Application Server administrative Java APIs

This section describes how to develop a Java program for accessing the WebSphere Application Server administrative system by using the WebSphere Application Server administrative application programming interfaces (APIs).

This task assumes a basic familiarity with Java Management Extensions (JMX) API programming. See the JMX API documentation for information.

When you develop and run administrative clients that use various JMX connectors and that have security enabled, use the following guidelines. When you follow these guidelines, you guarantee the behavior among different implementations of JMX connectors. Any programming model that strays from these guidelines is unsupported.

1. Create and use a single administrative client before you create and use another administrative client.
2. Create and use an administrative client on the same thread.
3. Use one of the following ways to specify a user ID and password to create a new administrative client:
   - Specify a default user ID and password in the property file.
   - Specify a user ID and password other than the default. Once you create an administrative client with a non-default user ID and password, specify the non-default user ID and password when you create subsequent administrative clients.

Develop an administrative client program.

Developing an administrative client program

This page contains examples of key features of an administrative client program that utilizes WebSphere Application Server administrative APIs and Java Management Extensions (JMX). WebSphere Application Server administrative APIs provide control of the operational aspects of your distributed system as well as the ability to update your configuration. This page also demonstrates examples of operational control. For information, view the Administrative API documentation, the JMX API documentation, or the MBean API documentation.

1. Create an AdminClient instance. An administrative client program needs to invoke methods on the AdminService object that is running in the deployment manager (or the application server in the base installation). The AdminClient class provides a proxy to the remote AdminService object through one of the supported Java Management Extensions (JMX) connectors. The following example shows how to create an AdminClient instance:

```
Properties connectProps = new Properties();
connectProps.setProperty(
AdminClient.CONNECTOR_TYPE, AdminClient.CONNECTOR_TYPE_SOAP);

connectProps.setProperty(AdminClient.CONNECTOR_HOST, "localhost");
connectProps.setProperty(AdminClient.CONNECTOR_PORT, "8879");
AdminClient adminClient = null;
try
{
       adminClient = AdminClientFactory.createAdminClient(connectProps);
}
catch (ConnectorException e)
{
       System.out.println("Exception creating admin client: " + e);
}
```

2. Find an MBean Once you obtain an AdminClient instance, you can use it to access managed resources in the administration servers and application servers. Each managed resource registers an MBean with the AdminService through which you can access the resource. The MBean is represented by an ObjectName instance that identifies the MBean. An ObjectName consists of a domain name followed by an unordered set of one or more key properties. For the WebSphere Application Server, the domain name is `WebSphere` and the key properties defined for administration are as follows:

type	The type of MBean. For example: Server, TraceService, Java Virtual Machine (JVM).
name	The name identifier for the individual instance of the MBean.
cell	The name of the cell that the MBean is running.
node	The name of the node that the MBean is running.

process	The name of the process that the MBean is running.

You can locate an MBean by querying for them with ObjectNames that match desired key properties. The following example shows how to find the MBean for the NodeAgent of node MyNode:

```
String nodeName = "MyNode";
String query = "WebSphere:type=NodeAgent,node=" + nodeName + ",*";
ObjectName queryName = new ObjectName(query);
ObjectName nodeAgent = null;
Set s = adminClient.queryNames(queryName, null);
if (!s.isEmpty())
    nodeAgent = (ObjectName)s.iterator().next();
else
    System.out.println("Node agent MBean was not found");
```

3. Use the MBean. What a particular MBean allows you to do depends on that MBean's management interface. It may declare attributes that you can obtain or set. It may declare operations that you can invoke. It may declare notifications for which you can register listeners. For the MBeans provided by the WebSphere Application Server, you can find information about the interfaces they support in the MBean API documentation. The following example invokes one of the operations available on the NodeAgent MBean that we located above. The following example will start the *MyServer* application server:

```
String opName = "launchProcess";
String signature[] = { "java.lang.String" };
String params[] = { "MyServer" };
try
{
     adminClient.invoke(nodeAgent, opName, params, signature);
}
catch (Exception e)
{
     System.out.println("Exception invoking launchProcess: " + e);
}
```

4. Register for events. In addition to managing resources, the Java Management Extensions (JMX) API also supports application monitoring for specific administrative events. Certain events produce notifications, for example, when a server starts. Administrative applications can register as listeners for these notifications. The WebSphere Application Server provides a full implementation of the JMX notification model, and provides additional function so you can receive notifications in a distributed environment. For a complete list of the notifications emitted from WebSphere Application Server MBeans, refer to the `com.ibm.websphere.management.NotificationConstants` class in the API documentation. The following is an example of how an object can register itself for event notifications emitted from an MBean using the node agent ObjectName:

```
adminClient.addNotificationListener(nodeAgent, this, null, null);
```

In this example, the `null` value will result in receiving all of the node agent MBean event notifications. You can also use the `null` value with the `handback` object.

5. Handle the events. Objects receive JMX event notifications via the handleNotification method which is defined by the NotificationListener interface and which any event receiver must implement. The following example is an implementation of handleNotification that reports the notifications that it receives:

```
public void handleNotification(Notification n, Object handback)
{
     System.out.println("***************************************************");
     System.out.println("* Notification received at " + new Date().toString());
     System.out.println("* type      = " + ntfyObj.getType());
     System.out.println("* message   = " + ntfyObj.getMessage());
     System.out.println("* source    = " + ntfyObj.getSource());
     System.out.println(
     "* seqNum    = " + Long.toString(ntfyObj.getSequenceNumber()));
```

```
        System.out.println("* timeStamp = " + new Date(ntfyObj.getTimeStamp()));
        System.out.println("* userData  = " + ntfyObj.getUserData());
        System.out.println("***************************************************");
    }
```

Administrative client program example: The following example is a complete administrative client program. Copy the contents to a file named `MyAdminClient.java`. After changing the node name and server name to the appropriate values for your configuration, you can compile and run it using the instructions from ″Creating a custom Java administrative client program using WebSphere Application Server administrative Java APIs″ in the information center.

```
import java.util.Date;
import java.util.Properties;
import java.util.Set;

import javax.management.InstanceNotFoundException;
import javax.management.MalformedObjectNameException;
import javax.management.Notification;
import javax.management.NotificationListener;
import javax.management.ObjectName;

import com.ibm.websphere.management.AdminClient;
import com.ibm.websphere.management.AdminClientFactory;
import com.ibm.websphere.management.exception.ConnectorException;

public class AdminClientExample implements NotificationListener
{

    private AdminClient adminClient;
    private ObjectName nodeAgent;
    private long ntfyCount = 0;

    public static void main(String[] args)
    {
       AdminClientExample ace = new AdminClientExample();

       // Create an AdminClient
       ace.createAdminClient();

       // Find a NodeAgent MBean
       ace.getNodeAgentMBean("ellington");

       // Invoke launchProcess
       ace.invokeLaunchProcess("server1");

       // Register for NodeAgent events
       ace.registerNotificationListener();

       // Run until interrupted
       ace.countNotifications();
    }

    private void createAdminClient()
    {
        // Set up a Properties object for the JMX connector attributes
        Properties connectProps = new Properties();
        connectProps.setProperty(
        AdminClient.CONNECTOR_TYPE, AdminClient.CONNECTOR_TYPE_SOAP);
        connectProps.setProperty(AdminClient.CONNECTOR_HOST, "localhost");
        connectProps.setProperty(AdminClient.CONNECTOR_PORT, "8879");

        // Get an AdminClient based on the connector properties
        try
        {
            adminClient = AdminClientFactory.createAdminClient(connectProps);
        }
        catch (ConnectorException e)
        {
            System.out.println("Exception creating admin client: " + e);
            System.exit(-1);
        }

        System.out.println("Connected to DeploymentManager");
    }


    private void getNodeAgentMBean(String nodeName)
    {
        // Query for the ObjectName of the NodeAgent MBean on the given node
        try
        {
            String query = "WebSphere:type=NodeAgent,node=" + nodeName + ",*";
            ObjectName queryName = new ObjectName(query);
            Set s = adminClient.queryNames(queryName, null);
            if (!s.isEmpty())
                nodeAgent = (ObjectName)s.iterator().next();
            else
            {
                System.out.println("Node agent MBean was not found");
                System.exit(-1);
            }
        }
        catch (MalformedObjectNameException e)
        {
            System.out.println(e);
            System.exit(-1);
        }
        catch (ConnectorException e)
        {
            System.out.println(e);
            System.exit(-1);
        }

        System.out.println("Found NodeAgent MBean for node " + nodeName);
    }

    private void invokeLaunchProcess(String serverName)
    {
        // Use the launchProcess operation on the NodeAgent MBean to start
        // the given server
        String opName = "launchProcess";
        String signature[] = { "java.lang.String" };
        String params[] = { serverName };
        boolean launched = false;
        try
        {
            Boolean b = (Boolean)adminClient.invoke(

nodeAgent, opName, params, signature);
            launched = b.booleanValue();
            if (launched)
                System.out.println(serverName + " was launched");
            else
                System.out.println(serverName + " was not launched");

        }
        catch (Exception e)
        {
            System.out.println("Exception invoking launchProcess: " + e);
        }
    }

    private void registerNotificationListener()
    {
        // Register this object as a listener for notifications from the
        // NodeAgent MBean.  Don't use a filter and don't use a handback
        // object.
        try
        {
            adminClient.addNotificationListener(nodeAgent, this, null, null);
            System.out.println("Registered for event notifications");
        }
        catch (InstanceNotFoundException e)
        {
            System.out.println(e);
        }
        catch (ConnectorException e)
        {
            System.out.println(e);
        }
    }

    public void handleNotification(Notification ntfyObj, Object handback)
    {
        // Each notification that the NodeAgent MBean generates will result in
        // this method being called
        ntfyCount++;
        System.out.println("***************************************************");
        System.out.println("* Notification received at " + new Date().toString());
        System.out.println("* type      = " + ntfyObj.getType());
        System.out.println("* message   = " + ntfyObj.getMessage());
        System.out.println("* source    = " + ntfyObj.getSource());
        System.out.println(
        "* seqNum    = " + Long.toString(ntfyObj.getSequenceNumber()));
        System.out.println("* timeStamp = " + new Date(ntfyObj.getTimeStamp()));
        System.out.println("* userData  = " + ntfyObj.getUserData());
        System.out.println("***************************************************");

    }

    private void countNotifications()
    {
        // Run until killed
        try
        {
            while (true)
            {
                Thread.currentThread().sleep(60000);
                System.out.println(ntfyCount + " notification have been received");
            }
        }
        catch (InterruptedException e)
        {
        }
    }

}
```

Extending the WebSphere Application Server administrative system with custom MBeans

You can extend the WebSphere Application Server administration system by supplying and registering new Java Management Extensions (JMX) MBeans (see JMX 1.0 Specification for details) in one of the WebSphere processes. JMX MBeans represent the management interface for a particular piece of logic. All of the managed resources within the standard WebSphere infrastructure are represented as JMX MBeans. There are a variety of ways in which you can create your own MBeans and register them with the JMX MBeanServer running in any WebSphere process. For more information, view the MBean API documentation.

1. Create custom JMX MBeans.

   You have some alternatives to select from, when creating MBeans to extend the WebSphere administrative system. You can use any existing JMX MBean from another application. You can register any MBean that you tested in a JMX MBean server outside of the WebSphere Application Server environment in a WebSphere Application Server process, including standard MBeans, dynamic MBeans, open MBeans, and model MBeans.

   In addition to any existing JMX MBeans, and ones that were written and tested outside of the WebSphere Application Server environment, you can use the special distributed extensions provided by WebSphere and create a WebSphere ExtensionMBean provider. This alternative provides better integration with all of the distributed functions of the WebSphere administrative system. An ExtensionMBean provider implies that you supply an XML file that contains an MBean Descriptor based on the DTD shipped with the WebSphere Application Server. This descriptor tells the WebSphere system all of the attributes, operations, and notifications that your MBean supports. With this information, the WebSphere system can route remote requests to your MBean and register remote Listeners to receive your MBean event notifications.

   All of the internal WebSphere MBeans follow the Model MBean pattern (see WebSphere Application Server administrative MBean documentation for details). Pure Java classes supply the real logic for management functions, and the WebSphere MBeanFactory class reads the description of these functions from the XML MBean Descriptor and creates an instance of a ModelMBean that matches the descriptor. This ModelMBean instance is bound to your Java classes and registered with the MBeanServer running in the same process as your classes. Your Java code now becomes callable from any WebSphere Application Server administrative client through the ModelMBean created and registered to represent it.

2. Register the new MBeans. There are various ways to register your MBean.

   You can register your MBean with the WebSphere Application Server administrative service.

   You can register your MBean with the MBeanServer in a WebSphere Application Server process. The following list describes the available options in order of preference:

   - Go through the MBeanFactory class. If you want the greatest possible integration with the WebSphere Application Server system, then use the MBeanFactory class to manage the life cycle of your MBean through the activateMBean and deactivateMBean methods of the MBeanFactory class. Use these methods, by supplying a subclass of the RuntimeCollaborator abstract superclass and an XML MBean descriptor file. Using this approach, you supply a pure Java class that implements the management interface defined in the MBean descriptor. The MBeanFactory class creates the actual ModelMBean and registers it with the WebSphere Application Server administrative system on your behalf.

     This option is recommended for registering model MBeans.
   - Use the JMXManageable and CustomService interface. You can make the process of integrating with WebSphere administration even easier, by implementing a CustomService interface, that also implements the JMXManageable interface. Using this approach, you can avoid supplying the RuntimeCollaborator. When your CustomService interface is initialized, the WebSphere MBeanFactory class reads your XML MBean descriptor file and creates, binds, and registers an MBean to your CustomService interface automatically. After the shutdown method of your CustomService is called, the WebSphere Application Server system automatically deactivates your MBean.
   - Go through the AdminService interface. You can call the registerMBean() method on the AdminService interface and the invocation is delegated to the underlying MBeanServer for the process, after appropriate security checks. You can obtain a reference to the AdminService using the getAdminService() method of the AdminServiceFactory class.

     This option is recommended for registering standard, dynamic, and open MBeans. Implement the UserCollaborator class to use the MBeans and to provide a consistent level of support for them across distributed and z/OS platforms.
   - Get MBeanServer instances directly. You can get a direct reference to the JMX MBeanServer instance running in any WebSphere Application Server process, by calling the getMBeanServer() method of the MBeanFactory class. You get a reference to the MBeanFactory class by calling the

getMBeanFactory() method of the AdminService interface. Registering the MBean directly with the MBeanServer instance can result in that MBean not participating fully in the distributed features of the WebSphere Application Server administrative system.

Regardless of the approach used to create and register your MBean, you must set up proper Java 2 security permissions for your new MBean code. The WebSphere AdminService and MBeanServer are tightly protected using Java 2 security permissions and if you do not explicitly grant your code base permissions, security exceptions are thrown when you attempt to invoke methods of these classes. If you are supplying your MBean as part of your application, you can set the permissions in the `was.policy` file that you supply as part of your application metadata. If you are using a CustomService interface or other code that is not delivered as an application, you can edit the `library.policy` file in the node configuration, or even the `server.policy` file in the `properties` directory for a specific installation.

Best practices for standard, dynamic, and open MBeans

This article discusses recommended guidelines for standard, dynamic, and open MBeans.

The underlying interface for the WebSphere Application Server administrative service is AdminService. Remote access occurs through the AdminControl scripting object.

For WebSphere Application Server Version 5, the MBean registration and capabilities are as follows:

MBean type	Registered with:	Capabilities
Model	WebSphere Application Server administrative service	Local access is through the WebSphere Application Server administrative service or the MBean server. Remote access is through the WebSphere Application Server administrative service, and WebSphere Application Server security.
Standard, dynamic, or open	MBean server	Local access is through the WebSphere Application Server administrative service or the MBean server on the distributed platform.

For V6, you can optionally register standard, dynamic, and open custom MBeans with the WebSphere Application Server administrative service to take advantage of the capabilities that in V5 are available only to model MBeans.

V6 introduces a special run-time collaborator that you use with standard, dynamic or open custom MBeans to register the custom MBeans with the WebSphere Application Server administrative service. The standard, dynamic, and open MBeans display in the administrative service as model MBeans. The administrative service uses the capabilities available to MBeans that are registered with the administrative service.

For WebSphere Application Server Version 6, the MBean registration and capabilities are as follows:

MBean type	Registered with:	Capabilities
Model, and optionally standard, dynamic, or open	WebSphere Application Server administrative service	Local access is through the WebSphere Application Server administrative service or the MBean server. Remote access is through the WebSphere Application Server administrative service, and WebSphere Application Server security.

Standard, dynamic, or open	MBean server	Local access is through the WebSphere Application Server administrative service or the MBean server on the distributed platform.

Creating and registering standard, dynamic, and open custom MBeans

You can create standard, dynamic, and open custom MBeans and register them with the WebSphere Application Server administrative service.

This task assumes a basic familiarity with MBean programming. For information on MBean programming see MBean Java application programming interface (API) documentation.

Perform the following tasks to create and register a standard, dynamic, or open custom MBean.

1. Create your particular MBean class or classes.
2. Write an MBean descriptor in the XML language for your MBean.
3. Register your MBean by inserting code that uses the WebSphere Application Server run-time com.ibm.websphere.management.UserMBeanCollaborator collaborator class into your application code.
4. Package the class files for your MBean interface and implementation, the descriptor XML file, and your application Java archive (JAR) file.

After you successfully complete the steps, you have a standard, dynamic, or open custom MBean that is registered and activated with the WebSphere Application Server administrative service.

The following example shows how to create and register a standard MBean with the WebSphere Application Server administrative service:

```
SnoopMBean.java:

/**
 * Use the SnoopMBean MBean, which has a standard mbean interface.
 */
public interface SnoopMBean {
    public String getIdentification();
    public void snoopy(String parm1);
}

SnoopMBeanImpl.java:

/**
 * SnoopMBeanImpl - SnoopMBean implementation
 */
public class SnoopMBeanImpl implements SnoopMBean {
    public String getIdentification() {
        System.out.println(">>> getIdentification() called...");
        return "snoopy!";
    }

    public void snoopy(String parm1) {
        System.out.println(">>> snoopy(" + parm1 + ") called...");
    }
}
```

Define the following MBean descriptor for your MBean in an .xml file. The getIdentification method is set to run with the unicall option and the snoopy method is set to use the multicall option. These options are used only for z/OS platform applications. The WebSphere Application Server for z/OS options are not applicable to the distributed platforms, but they do not need to be removed. The options are ignored on the distributed platforms.

SnoopMBean.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE MBean SYSTEM "MbeanDescriptor.dtd">
<MBean type="SnoopMBean"
 version="5.0"
 platform="dynamicproxy"
 description="Sample SnoopMBean to be initialized inside an EJB.">


 <attribute name="identification" getMethod="getIdentification"

       type="java.lang.String" proxyInvokeType="unicall"/>

 <operation name="snoopy" role="operation"  type="void" targetObjectType="objectReference"
impact="ACTION" proxyInvokeType="multicall">
  <signature>
   <parameter name="parm1" description="test parameter" type="java.lang.String"/>
  </signature>
 </operation>
</MBean>
```

Assume that your MBean is used in an enterprise bean. Register your MBean in the enterprise bean ejbCreate method and unregister it in the ejbRemove method.

```
//The method MBeanFactory.activateMBean() requires four parameters:
//String type: The type value that you put in this MBean's descriptor. For this example
//the string type is SnoopMBean.
//RuntimeCollaborator co: The UserMBeanCollaborator user MBean collaborator instance
//that you create
//String id: Unique name that you pick
//String desciptor: The MBean descriptor file name


import com.ibm.websphere.management.UserMBeanCollaborator;
//Import other classes here.
.
.
.
static private ObjectName snoopyON = null;
static private Object lockObj = "this is a lock";
.
.
.
/**
 * ejbCreate method: Register your Mbean.
 */
public void ejbCreate() throws javax.ejb.CreateException {
    synchronized (lockObj) {
        System.out.println(">>> SnoopMBean activating for --|" + this + "|--");
        if (snoopyON != null) {
            return;
        }
        try {
            System.out.println(">>> SnoopMBean activating...");
            MBeanFactory mbfactory = AdminServiceFactory.getMBeanFactory();
            RuntimeCollaborator snoop = new UserMBeanCollaborator(new SnoopMBeanImpl());
            snoopyON = mbfactory.activateMBean("SnoopMBean", snoop, "snoopMBeanId", "SnoopMBean.xml");
            System.out.println(">>> SnoopMBean activation COMPLETED! --|" + snoopyON + "|--");
        } catch (Exception e) {
            System.out.println(">>> SnoopMBean activation FAILED:");
            e.printStackTrace();
        }
    }
}
.
```

```
.
.
/**
 * ejbRemove method: Unregister your MBean.
 */
public void ejbRemove() {
    synchronized (lockObj) {
        System.out.println(">>> SnoopMBean Deactivating for --|" + this + "|--");
        if (snoopyON == null) {
            return;
        }
        try {
            System.out.println(">>> SnoopMBean Deactivating ==|" + snoopyON +
                "|== for --|" + this + "|--");
            MBeanFactory mbfactory = AdminServiceFactory.getMBeanFactory();
            mbfactory.deactivateMBean(snoopyON);
            System.out.println(">>> SnoopMBean Deactivation COMPLETED!");
        } catch (Exception e) {
            System.out.println(">>> SnoopMBean Deactivation FAILED:");
            e.printStackTrace();
        }
    }
}
```

Compile the MBean Java files and package the resulting class files with the descriptor .xml file, into the enterprise bean JAR file.

Java 2 security permissions

When you enable Java 2 security, you must grant Java 2 security permissions to application-specific code for Java Management Extensions (JMX) and WebSphere Application Server administrative privileges. With these permissions, your application code can call WebSphere Application Server administrative methods and JMX methods.

Use the following permission to invoke all the JMX class methods and interface methods:

```
permission javax.management.MBeanPermission "*", "*";
```

Consult the Java Management Extensions (JMX) API documentation for specific actions under the MBeanPermission class.

Use the following permission for WebSphere Application Server administrative application programming interfaces (APIs):

```
permission com.ibm.websphere.security.WebSphereRuntimePermission "AdminPermission";
```

Developing administrative programs for multiple Java 2 Platform, Enterprise Edition application servers

You can develop an administrative client to manage multiple vendor application servers through existing MBean support in the WebSphere Application Server.

Existence of MBeans for stopped components

The WebSphere Application Server completely implements the Java 2 Platform, Enterprise Edition (J2EE) Management specification. However, some differences in details between the J2EE specification and the WebSphere Application Server implementation are important for you to understand when you access WebSphere Application Server components. These differences are important to you when you access application MBeans because you can use either the WebSphere Application Server programming model or the J2EE programming model.

In the WebSphere Application Server programming model, if an MBean exists, you can assume that it is running. If an MBean does not exist, you can assume that it is stopped. Transient states between the started state and the stopped state are the same as the stopped state, which means that no MBean exists.

In the J2EE programming model, the MBean always exists regardless of the state of the component.

You can determine the state of a component by querying the state attribute. However, the state attribute only exists for MBeans that are state manageable, meaning that they implement the StateManageable interface. State manageable MBeans have start(), startRecursive(), and stop() operations whether these MBeans are J2EE MBeans or WebSphere Application Server MBeans. Additionally, the WebSphere Application Server defines the stateful interface. The stateful interface means that the component has a state and emits the j2ee.state.notifications method, but that the component cannot directly manage the state. For example, a Web module cannot stop itself. However, the application that contains the Web module can stop it.

Not all MBeans that have a state are state-manageable. Servlets, J2EE modules and enterprise beans, for example, are all stateful, but are not state manageable. The J2EE server is not state-manageable because no start() operation is available on a server.

The J2EEApplication MBean is an example of a state manageable MBean. When the WebSphere Application Server starts, each application activates a J2EEApplication MBean for itself. A J2EEApplication MBean has a J2EE type of J2EEApplication (for example, `ObjectName *:*,j2eeType=J2EEApplication`). If the application starts, it also activates an Application MBean with a type of Application (for example, `*:*,type=Application`). When the application changes state, the Application MBean is activated or deactivated. However, the J2EEApplication MBean is always activated. You can retrieve the application state changes by getting the state attribute.

The modules attribute on the J2EEApplication component returns an array of object names, one for every module in the application. The Application Server activates an MBean for each of these modules only after the Application Server starts the application. The managed enterprise bean isRegistered(ObjectName) method returns `false` if the application, and therefore the module, is not running.

All of the attributes that are defined in the J2EE management specification return valid values when the managed object stops. Other attributes and operations, for example those that are specifically defined for the Application Server, use the com.ibm.websphere.management.exception.ObjectNotRunningException exception if they are accessed when the object is stopped.

If you install the application while the server runs, the application installs the J2EEApplication MBean when the installation completes. Conversely, when the application uninstalls the J2EEApplication MBean, the application deactivates the MBean.

Mapping key properties

The following table lists the mapping from the J2EE management-defined j2eeType key property to the WebSphere Application Server type key property. You can use either key property to access MBeans. However, only use the j2eeType key property if you want to connect to application servers other than WebSphere Application Server.

j2eeType key property	type key property
J2EEDomain	J2EEDomain [new]
J2EEServer	Server
JVM	JVM
J2EEApplication	Application [separate MBean from j2eeType

WebModule	WebModule
ResourceAdapterModule	ResourceAdapterModule
EJBModule	EJBModule
EJB and subtypes / Servlet / ResourceAdapter	EJB and subtypes / Servlet / ResourceAdapter
JavaMailResource	MailProvider
JNDIResource	NameServer
JMSResource	JMSProvider
JTAResource	TransactionService
RMI_IIOPResource	ORB
URLResource	URLProvider
JDBCResource	JDBCProvider
JDBCDataSource	DataSource
JDBCDriver	JDBCDriver [new]
JCAResource	J2CResourceAdapter
JCAConnectionFactory	J2CConnectionFactory
JCAManagedConnectionFactory	J2CManagedConnectionFactory [new]

Optional WebSphere Application Server interfaces

The following table shows the optional J2EE management interfaces that WebSphere Application Server provides.

j2eeType key property	EventProvider interface	StateManageable interface	StatisticsProvider interface
J2EEDomain	No	No	No
J2EEServer	Yes	Stateful	No
JVM	No	No	Yes
J2EEApplication	Yes	Yes	No
WebModule	Yes	Stateful	No
ResourceAdapterModule	Yes	Stateful	No
EJBModule	Yes	Stateful	No
AppClientModule	Yes	Stateful	No
EJB and subtypes / Servlet / ResourceAdapter	No	No	Yes
JavaMailResource	No	No	No
JNDIResource	No	No	No
JMSResource	No	No	No
JTAResource	Yes	No	Yes
RMI_IIOPResource	No	No	Yes
URLResource	No	No	No
JDBCResource	No	No	Yes
JDBCDataSource	No	No	No
JDBCDriver	No	No	No

JCAResource	Yes	No	Yes
JCAConnectionFactory	No	No	No
JCAManagedConnectionFactory	No	No	No

Deploying and managing a custom Java administrative client program with multiple Java 2 Platform, Enterprise Edition application servers

This section describes how to connect to a Java 2 Platform, Enterprise Edition (J2EE) server, and how to manage multiple vendor servers.

The WebSphere Application Server completely implements the J2EE Management specification, also known as JSR-77 (Java Specification Requests 77). However, some differences in details between the J2EE specification and the WebSphere Application Server implementation are important for you to understand when you develop a Java administrative client program to manage multiple vendor servers. For information, see the Java Platform, Enterprise Edition (J2EE) Management Specification and the MBean application programming interface (API) documentation.

When your administrative client program accesses WebSphere Application Servers exclusively, you can use the Java APIs and WebSphere Application Server-defined MBeans to manage them. If your program needs to access both WebSphere Application Servers and other J2EE servers, use the API defined in the J2EE Management specification.

1. Connect to a J2EE server.

   Connect to a server by looking up the Management enterprise bean from the Java Naming and Directory Interface (JNDI). The Management enterprise bean supplies a remote interface to the MBean server that runs in the application server. The Management enterprise bean works almost exactly like the WebSphere Application Server administrative client, except that it does not provide WebSphere Application Server specific functionality. The following example shows how to look up the Management enterprise bean.

   ```
   import javax.management.j2ee.ManagementHome;
   import javax.management.j2ee.Management;

   Properties props = new Properties();

   props.setProperty(Context.PROVIDER_URL, "iiop://myhost:2809");
   Context ic = new InitialContext(props);
   Object obj = ic.lookup("ejb/mgmt/MEJB");
   ManagementHome mejbHome = (ManagementHome)
           PortableRemoteObject.narrow(obj, ManagementHome.class);
   Management mejb = mejbHome.create();
   ```

   The example gets an initial context to an application server by passing the host and port of the Remote Method Invocation (RMI) connector. You must explicitly code the RMI port, in this case 2809. The lookup method looks up the `ejb/mgmt/MEJB` path, which is the location of the Management enterprise bean home. The example then creates the mejb stateless session bean, which you use in the next step.

2. Manage multiple vendor application servers.

   After you create the mejb stateless session bean, you can use it to manage your application servers. Components from the application servers appear as MBeans, which the specification defines. These MBeans all have the j2eeType key property. This key property is one of a set of types that the specification defines. All of these types have a set of exposed attributes.

   Use the following example to guide you in managing multiple vendor application servers. The example uses the Java virtual machine (JVM) MBean to determine what the current heap size is for the application server.

```
ObjectName jvmQuery = new ObjectName("*:j2eeType=JVM,*");
Set s = mejb.queryNames(jvmQuery, null);
ObjectName jvmMBean = (ObjectName) s.iterator().next();
boolean hasStats = ((Boolean) mejb.getAttribute(jvmMBean,
        "statisticsProvider")).booleanValue();
if (hasStats) {
    JVMStats stats = (JVMStats) mejb.getAttribute(jvmMBean,
                                                  "stats");
    String[] statisticNames = stats.getStatisticNames();
    if (Arrays.asList(statisticNames).contains("heapSize")) {
        System.out.println("Heap size: " + stats.getHeapSize());
    }
}
```

The queryNames() method first queries the JVM MBean. The getAttribute method gets the statisticsProvider attribute and determine if this MBean provides statistics. If the MBean does, the example accesses the stats attribute, and then invokes the getHeapSize() method to get the heap size.

The strength of this example is that the example can run on any vendor application server. It demonstrates that an MBean can optionally implement defined interfaces, in this case the StatisticsProvider interface. If an MBean implements the StatisticsProvider interface, you can see if an application server supports a particular statistic, in this case the heap size. The specification defines the heap size, although this value is optional. If the application server supports the heap size, you can display the heap size for the JVM.

Migrating Java Management Extensions V1.0 to Java Management Extensions V1.2

Each Java Virtual Machine (JVM) in WebSphere Application Server includes an embedded implementation of Java Management Extensions (JMX). In Application Server, Version 5, the JVMs contain an implementation of the JMX 1.0 specification. In Application Server, Version 6, the JVMs contain an implementation of the JMX 1.2 specification. The JMX 1.0 implementation used in Version 5 is the TMX4J package that IBM Tivoli products supply. The JMX 1.2 specification used in Version 6 is the open source mx4j package. The JMX implementation change across the releases does not affect the behavior of the JMX MBeans in the Application Server. No Application Server administrative application programming interfaces (APIs) are altered due to the change from the JMX V1.0 specification to the JMX V1.2 specification.

The JMX V1.2 specification is backward compatible with the JMX 1.0 specification. However, you might need to migrate custom MBeans that are supplied by products other than the Application Server from Version 5 to Version 6. The primary concern for these custom MBeans is related to the values that are used in key properties of the JMX ObjectName class for the MBean. The open source mx4j implementation more stringently enforces property validation according to the JMX 1.2 specification. Test the custom MBeans that you deployed in Version 5 in Version 6, to ensure compatibility. Full details of the JMX V1.2 specification changes from the JMX V1.0 specification are available in the JMX 1.2 specification.

Java Management Extensions interoperability

WebSphere Application Server Version 6 implements Java Management Extensions (JMX) Version 1.2, while WebSphere Application Server Version 5 implements JMX Version 1.0.

Due to the evolution of the JMX specification, the serialization format for JMX objects, such as the javax.management.ObjectName object, differs between the V5 implementation and the V6 implementation. The V6 JMX run time is enhanced to be aware of the version of the client with which it is communicating. The V6 run time makes appropriate transformations on these incompatible serialized formats to support communication between the different version run times. A V5 wsadmin script or a V5 administrative client can call a V6 deployment manager, node, or server. A V6 wsadmin script or a V6 administrative client can call a V5 node or server.

When a V5 wsadmin script or a V5 administrative client calls a V6 MBean, the instances of classes that are new in V6 cannot be passed back to V5 because these classes are not present in the V5 environment. The problem occurs infrequently. However, it usually occurs when an exception embeds a nested exception that is new in V6. The symptom is usually a serialization exception or a NoClassDefFoundException exception.

Due to changes in the JMX implementation from V5 to V6, different exceptions are created when a method on an MBean is invoked for V5 than when a method on an MBean is invoked for V6. For example, when a method gets or sets an unknown attribute for V5, the MBeanRuntimeException exception is created. When a method gets or sets an unknown attribute for V6, the MBeanException exception that wraps a ServiceNotFoundException exception is created.

An instance of a user-defined class that implements the Serializable interface that is passed as a parameter or return value during MBean invocation, or sent as part of a notification, cannot contain a non-transient instance variable that is in the javax.management.package package. If the instance does, it cannot be properly deserialized when passed between V5 and V6 run times.

Due to changes in the supported format for the ObjectName class from V5 to V6, the configuration ID in V6 contains a vertical bar (|), whereas in V5, the ID contains a colon (:). This change is reflected in the output for wsadmin clients. For example, for a V5 client, the output is:

```
wsadmin>  $AdminConfig list Cell
     DefaultCellNetwork(cells/DefaultCellNetwork:cell.xml#Cell_1)
```

whereas for a V6 client, the output is:

```
wsadmin>  $AdminConfig list Cell
     DefaultCellNetwork(cells/DefaultCellNetwork|cell.xml#Cell_1)
```

The change to the configuration ID generally is not a problem because configuration IDs are generated dynamically. When a V5 client passes a configuration ID that contains a colon, the JMX run time, for upward compatibility, automatically transforms the configuration ID that contains a colon into a configuration ID that contains a vertical bar. Similarly, a reverse transformation is performed for backward compatibility.

Do not save the configuration ID and then try to use it later. Only query the ID and use it.

Managing applications through programming

This topic describes how, through Java MBean programming, to install, update, and delete a Java 2 Platform, Enterprise Edition (J2EE) application on WebSphere Application Server.

This task assumes a basic familiarity with MBean programming. For information on MBean programming see MBean Java application programming interface (API) documentation.

Before you can install or change an application on WebSphere Application Server, you must first create or update your application and assemble it using an assembly tool.

Besides installing, uninstalling, and updating applications through programming, you can additionally install, uninstall, and update J2EE applications through the administrative console or the wsadmin tool. All three ways provide identical updating capabilities.

1. Perform any or all of the following tasks to manage your J2EE applications through programming.
   a. Install an application.

      This article provides an example for initially installing an application on WebSphere Application Server.

   b. Uninstall an application.

This article provides an example for uninstalling an application that resides on WebSphere Application Server.

   c. Update an application.

      This article provides an example for updating the installed application on WebSphere Application Server with a new application. When you completely update an application, the deployed application is uninstalled and the new enterprise archive (EAR) file is installed.

   d. Add to, update, or delete part of an application.

      This article provides an example that you can use to add to, update, or delete part of an application on WebSphere Application Server.

   e. Add a module.

      This article provides an example for adding a module to an application that resides on WebSphere Application Server.

   f. Update a module.

      This article provides an example for updating a module that resides on WebSphere Application Server. When you update a module, the deployed module is uninstalled and the updated module is installed.

   g. Delete a module.

      This article provides an example for deleting a module that resides on WebSphere Application Server. When you delete a module, the deployed module is uninstalled.

   h. Add a file.

      This article provides an example for adding a file to an application that resides on WebSphere Application Server.

   i. Update a file.

      This article provides an example for updating a file on WebSphere Application Server. When you update a file, the deployed file is uninstalled and the updated file is installed.

   j. Delete a file.

      This article provides an example for deleting a file on WebSphere Application Server. When you delete a file, the deployed file is uninstalled.

2. Save your changes to the master configuration repository.

If you have further application updates, you can do the updates through programming, the administrative console, or the wsadmin tool.

Installing an application through programming

You can install an application through the administrative console, the wsadmin tool, or programming. Use this example to install an application through programming.

This task assumes a basic familiarity with MBean programming. For information on MBean programming see MBean Java application programming interface (API) documentation.

Before you can install an application on WebSphere Application Server, you must first create or update your application and assemble it using an assembly tool.

Perform the following tasks to install an application through programming.

1. Populate the enterprise archive (EAR) file with WebSphere Application Server-specific binding information.
   a. Create the controller and populate the EAR file with appropriate options.
   b. Optionally run the default binding generator.
   c. Save and close the EAR file.
   d. Retrieve the saved options table that will be passed to the installApplication MBean (API).

2. Connect to WebSphere Application Server.
3. Create the application management proxy.
4. If the preparation phase (population of the EAR file) is not performed, the do the following actions:
   a. Create an options table to be passed to the installApplication MBean API.
   b. Create a table for module to server relations and add the table to the options table.

      Refer to the com.ibm.websphere.management.application.AppManagement class in the Application Server API documentation to understand various options that can be passed to the installApplication MBean API.
5. Create the notification filter for listening to installation events.
6. Add the listener.
7. Install the application.
8. Wait for some timeout so that the program does not end.
9. Listen to Java Management Extensions (JMX) notifications to understand completion of the operation.
10. When the installation is done, remove the listener and quit.

After you successfully run the code, the application is installed.

The following example shows how to install an application based on the previous steps:

```
import java.lang.*;
import java.io.*;
import java.util.*;
import java.lang.reflect.*;
import com.ibm.websphere.management.application.*;
import com.ibm.websphere.management.application.client.*;
import com.ibm.websphere.management.*;

import javax.management.*;

public class Install {

    public static void main (String [] args) {

        try {
  String earFile = "C:/test/test.ear";
  String appName = "MyApp";

// Preparation phase: Begin
// Through the preparation phase you populate the enterprise archive (EAR) file with
// WebSphere Application Server-specific binding information. For example, you can specify
// Java Naming and Directory Interface (JNDI) names for enterprise beans, or virtual hosts
// for Web modules, and so on.

// First, create the controller and populate the EAR file with the appropriate options.
  Hashtable prefs = new Hashtable();
  prefs.put(AppConstants.APPDEPL_LOCALE, Locale.getDefault());

// You can optionally run the default binding generator by using the following options.
// Refer to Java documentation for the AppDeploymentController class to see all the
// options that you can set.
  Properties defaultBnd = new Properties();
  prefs.put (AppConstants.APPDEPL_DFLTBNDG, defaultBnd);
  defaultBnd.put (AppConstants.APPDEPL_DFLTBNDG_VHOST, "default_host");

// Create the controller.
  AppDeploymentController controller = AppDeploymentController
    .readArchive(earFile, prefs);
  AppDeploymentTask task = controller.getFirstTask();
  while (task != null)
  {
// Populate the task data.
```

```
   String[][] data = task.getTaskData();
// Manipulate task data which is a table of stringtask.
   setTaskData (data);
   task = controller.getNextTask();
  }
  controller.saveAndClose();

  Hashtable options = controller.getAppDeploymentSavedResults();
// The previous options table contains the module-to-server relationship if it was set by
// using tasks.
//Preparation phase: End

// Get a connection to WebSphere Application Server.
  String host = "localhost";
  String port = "8880";
  String target = "WebSphere:cell=cellName,node=nodeName,server=server1";

  Properties config = new Properties();
  config.put (AdminClient.CONNECTOR_HOST,  host);
  config.put (AdminClient.CONNECTOR_PORT,  port);
  config.put (AdminClient.CONNECTOR_TYPE, AdminClient.CONNECTOR_TYPE_SOAP);
  System.out.println ("Config: " + config);
     AdminClient _soapClient = AdminClientFactory.createAdminClient(config);

// Create the application management proxy, AppManagement.
  AppManagement proxy = AppManagementProxy. getJMXProxyForClient (_soapClient);

// If code for the preparation phase has been run, then you already have the options table.
// If not, create a new table and add the module-to-server relationship to it by uncommenting
// the next statement.
//Hashtable options = new Hashtable();
  options.put (AppConstants.APPDEPL_LOCALE, Locale.getDefault());

// Uncomment the following statements to add the module to the server relationship table if
//  the preparation phase does not collect it.
//Hashtable module2server = new Hashtable();
//module2server.put ("*", target);
//options.put (AppConstants.APPDEPL_MODULE_TO_SERVER, module2server);

//Create the notification filter for listening to installation events.
  NotificationFilterSupport myFilter = new NotificationFilterSupport();
  myFilter.enableType (AppConstants.NotificationType);


//Add the listener.
  NotificationListener listener = new AListener(_soapClient, myFilter,
      "Install: " + appName, AppNotification.INSTALL);

// Install the application.
  proxy.installApplication (earFile, appName, options, null);
  System.out.println ("After install App is called..");

// Wait for some timeout. The installation application programming interface (API) is
//  asynchronous and so returns immediately.
// If the program does not wait here, the program ends.
  Thread.sleep(300000); // Wait so that the program does not end.

        }
        catch (Exception e) {
            e.printStackTrace();
        }

    }

}

// Specify the Java Management Extensions (JMX) notification listener for JMX events.
```

```
class AListener implements NotificationListener
{
    AdminClient _soapClient;
    NotificationFilterSupport  myFilter;
    Object handback;
    ObjectName on;
    String eventTypeToCheck;

    public AListener(AdminClient cl, NotificationFilterSupport fl, Object h,
        String eType) throws Exception
    {
        _soapClient = cl;
        myFilter = fl;
        handback = h;
        eventTypeToCheck = eType;

        Iterator iter = _soapClient.queryNames (new ObjectName("WebSphere:type=
            AppManagement,*"), null).iterator();
        on = (ObjectName)iter.next();
        System.out.println ("ObjectName: " + on);
        _soapClient.addNotificationListener (on, this, myFilter, handback);
    }

    public void handleNotification (Notification notf, Object handback)
    {
            AppNotification ev = (AppNotification) notf.getUserData();
            System.out.println ("!! JMX event Recd: (handback obj= " + handback+ "): " + ev);


            //When the installation is done, remove the listener and quit.

            if (ev.taskName.equals (eventTypeToCheck) &&
                (ev.taskStatus.equals (AppNotification.STATUS_COMPLETED) ||
                 ev.taskStatus.equals (AppNotification.STATUS_FAILED)))
            {
                    try
                    {
                            _soapClient.removeNotificationListener (on, this);
                    }
            catch (Throwable th)
                    {
                        System.out.println ("Error removing listener: " + th);
                    }
                    System.exit (0);
        }
    }
}
```

Once you install the application, you must explicitly start the application or restart the server.

Starting an application through programming:

You can start an application through the administrative console, the wsadmin tool, or programming. Use this example to start an application through programming.

This task assumes a basic familiarity with MBean programming. For information on MBean programming see MBean Java application programming interface (API) documentation.

Before you can start an application on WebSphere Application Server, you must first install your application.

Perform the following tasks to start an application through programming.

1. Connect the administrative client to WebSphere Application Server.

2. Create the application management proxy.
3. Call the startApplication method on the proxy by passing the application name and optionally the list of targets on which to start the application.

After you successfully run the code, the application is started.

The following example shows how to start an application following the previously listed steps:

```
//Do a get of the administrative client to connect to
//WebSphere Application Server.

AdminClient client = ...;
String appName = "myApp";
Hashtable prefs = new Hashtable();
// Use the AppManagement MBean to start and stop applications on all or some targets.
// The AppManagement MBean is on the deployment manager in the Network Deployment product
// or on server1 in WebSphere Application Server.
// Query and get the AppManagement MBean.
ObjectName on = new ObjectName ("WebSphere:type=AppManagement,process=dmgr,*");
Iterator iter = client.queryNames (on, null).iterator();
ObjectName appmgmtON = (ObjectName)iter.next();

//Start the application on all targets.
AppManagement proxy = AppManagementProxy.getJMXProxyForClient(client);
String started = proxy.startApplication(appName, prefs, null);
System.out.println("Application started on folloing servers: " + started);

//Start the application on some targets.
//String targets = "WebSphere:cell=cellname,node=nodename,server=
       servername+WebSphere:cell=cellname,cluster=clusterName";
//String started1 = proxy.startApplication(appName, targets, prefs, null);
//System.out.println("Application started on following servers: " + started1)
```

Uninstalling an application through programming

You can uninstall an application through the administrative console, the wsadmin tool, or programming. Use this example to uninstall an application through programming.

This task assumes a basic familiarity with MBean programming. For information on MBean programming see MBean Java application programming interface (API) documentation.

Before you can uninstall an application on WebSphere Application Server, you must first install it.

Perform the following tasks to uninstall an application through programming.

1. Get a connection to WebSphere Application Server.
2. Get the application management proxy.
3. Create the notification filter for listening to uninstallation events.
4. Add the listener.
5. Uninstall the application.
6. Wait for some timeout so that the program does not end.
7. Listen to Java Management Extensions (JMX) notifications to understand completion of the operation.
8. When the uninstallation is done, remove the listener and quit.

After you successfully run the code, the application is uninstalled.

The following example shows how to uninstall an application based on the previous steps:

```
import java.lang.*;
import java.io.*;
import java.util.*;
import java.lang.reflect.*;
import com.ibm.websphere.management.application.*;
import com.ibm.websphere.management.application.client.*;
import com.ibm.websphere.management.*;

import javax.management.*;

public class Uninstall {

    public static void main (String [] args) {

        try {

// Get a connection to the server.
  String host = "localhost";
  String port = "8880";
  String target = "WebSphere:cell=cellName,node=nodeName,server=server1";

  Properties config = new Properties();
  config.put (AdminClient.CONNECTOR_HOST,  host);
  config.put (AdminClient.CONNECTOR_PORT,  port);
  config.put (AdminClient.CONNECTOR_TYPE, AdminClient.CONNECTOR_TYPE_SOAP);
  System.out.println ("Config: " + config);
     AdminClient _soapClient = AdminClientFactory.createAdminClient(config);

  // Get the application management proxy.
  AppManagement proxy = AppManagementProxy. getJMXProxyForClient (_soapClient);

  String appName = "MyApp";
  Hashtable options = new Hashtable();
  options.put (AppConstants.APPDEPL_LOCALE, Locale.getDefault());


  //Create the notification filter.
  NotificationFilterSupport myFilter = new NotificationFilterSupport();
  myFilter.enableType (AppConstants.NotificationType);


  //Add the listener.
  NotificationListener listener = new AListener(_soapClient, myFilter,
     "Install: " + appName, AppNotification.UNINSTALL);

  // Uninstall the application.
  proxy.uninstallApplication (appName, options, null);
  System.out.println ("After uninstall App is called..");

// Wait for some timeout. The installation application programming interface (API) is
//  asynchronous and so returns immediately.
// If the program does not wait here, the program ends.
  Thread.sleep(300000); // Wait so that the program does not end.

        }
        catch (Exception e) {
            e.printStackTrace();
        }

    }

}

// Specify the Java Management Extensions (JMX) notification listener for JMX events.
class AListener implements NotificationListener
{
    AdminClient _soapClient;
    NotificationFilterSupport  myFilter;
    Object handback;
    ObjectName on;
    String eventTypeToCheck;
    public AListener(AdminClient cl, NotificationFilterSupport fl, Object h,
          String eType) throws Exception
    {
        _soapClient = cl;
        myFilter = fl;
        handback = h;
        eventTypeToCheck = eType;

        Iterator iter = _soapClient.queryNames (new ObjectName("WebSphere:
            type=AppManagement,*"), null).iterator();
        on = (ObjectName)iter.next();
        System.out.println ("ObjectName: " + on);
        _soapClient.addNotificationListener (on, this, myFilter, handback);
    }

    public void handleNotification (Notification notf, Object handback)
    {
            AppNotification ev = (AppNotification) notf.getUserData();
            System.out.println ("!! JMX event Recd: (handback obj= " + handback+
               "): " + ev);


            //When the unistallation is done, remove the listener and quit

            if (ev.taskName.equals (eventTypeToCheck) &&
                (ev.taskStatus.equals (AppNotification.STATUS_COMPLETED) ||
                 ev.taskStatus.equals (AppNotification.STATUS_FAILED)))
            {
                    try
                    {
                            _soapClient.removeNotificationListener (on, this);
                    }
            catch (Throwable th)
                    {
                        System.out.println ("Error removing listener: " + th);
                    }
                    System.exit (0);
        }
    }
}
```

Updating an application through programming

You can update an existing application through the administrative console, the wsadmin tool, or programming. Use this example to completely update an application through programming.

This task assumes a basic familiarity with MBean programming. For information on MBean programming see MBean Java application programming interface (API) documentation.

Before you can update an application on WebSphere Application Server, you must first install your application.

Perform the following tasks to completely updaste an application through programming.

1. Connect to WebSphere Application Server.
2. Create the application management proxy.
3. Create the notification filter for listening to events.
4. Add the listener.
5. Prepare the enterprise archive (EAR) file by populating it with binding information.
6. Update the application.
7. Wait for some timeout so that the program does not end.
8. Listen to Java Management Extensions (JMX) notifications to understand completion of the operation.

9. When the update is done, remove the listener and quit.

After you successfully run the code, the application is updated.

The following example shows how to update an application based on the previous steps:

```
import java.lang.*;
import java.io.*;
import java.util.*;
import java.lang.reflect.*;
import com.ibm.websphere.management.application.*;
import com.ibm.websphere.management.application.client.*;
import com.ibm.websphere.management.*;

import javax.management.*;

public class aa {

    public static void main (String [] args) {

        try {

  // Connect to WebSphere Application Server.
  String host = "localhost";
  String port = "8880";
  String target = "WebSphere:cell=cellName,node=nodeName,server=server1";

  Properties config = new Properties();
  config.put (AdminClient.CONNECTOR_HOST,  host);
  config.put (AdminClient.CONNECTOR_PORT,  port);
  config.put (AdminClient.CONNECTOR_TYPE, AdminClient.CONNECTOR_TYPE_SOAP);
  System.out.println ("Config: " + config);
     AdminClient _soapClient = AdminClientFactory.createAdminClient(config);

  // Create the application management proxy, AppManagement.
  AppManagement proxy = AppManagementProxy. getJMXProxyForClient (_soapClient);

  String appName = "MyApp";
  String fileContents = "C:/test/test.ear";

  // Create the notification filter.
  NotificationFilterSupport myFilter = new NotificationFilterSupport();
  myFilter.enableType (NotificationConstants.TYPE_APPMANAGEMENT);
  //Add the listener.
  NotificationListener listener = new AListener(_soapClient, myFilter,
      "Install: " + appName, AppNotification.INSTALL);

// Refer to the installation example to see how you can prepare the enterprise
      archive (EAR)
// file by populating it with binding information.
// If code for the preparation phase has started, then you already have the options table.
// If not, create a new table and add the module-to-server relationship to it by uncommenting
// the next statement.
//Hashtable options = new Hashtable();
  options.put (AppConstants.APPDEPL_LOCALE, Locale.getDefault());
  options.put ((AppConstants.APPUPDATE_CONTENTTYPE, AppConstants.APPUPDATE_CONTENT_APP);

// Uncomment the following statements to add the module to the server relationship table if
//  the preparation phase does not collect it
//Hashtable module2server = new Hashtable();
//module2server.put ("*", target);
//options.put (AppConstants.APPDEPL_MODULE_TO_SERVER, module2server);
// Update the application.
  proxy.updateApplication (   appName,
                 null,
                 fileContents,
                 AppConstants.APPUPDATE_UPDATE,
                options,
                null);

// Wait for some timeout. The installation application programming interface (API) is
//  asynchronous and so returns immediately.
// If the program does not wait here, the program ends.
  Thread.sleep(300000); // Wait so that the program does not end.

        }
        catch (Exception e) {
            e.printStackTrace();
        }

    }

}

// Specify the Java Management Extensions (JMX) notification listener for JMX events.
class AListener implements NotificationListener
{
    AdminClient _soapClient;
    NotificationFilterSupport  myFilter;
    Object handback;
    ObjectName on;
    String eventTypeToCheck;

    public AListener(AdminClient cl, NotificationFilterSupport fl, Object h,
       String eType) throws Exception
    {
        _soapClient = cl;
        myFilter = fl;
        handback = h;
        eventTypeToCheck = eType;

        Iterator iter = _soapClient.queryNames (new ObjectName
             ("WebSphere:type=AppManagement,*"), null).iterator();
        on = (ObjectName)iter.next();
        System.out.println ("ObjectName: " + on);
        _soapClient.addNotificationListener (on, this, myFilter, handback);
    }

    public void handleNotification (Notification notf, Object handback)
    {
            AppNotification ev = (AppNotification) notf.getUserData();
            System.out.println ("!! JMX event Recd: (handback obj= " +
                 handback+ "): " + ev);


            //When the installation is done, remove the listener and quit

            if (ev.taskName.equals (eventTypeToCheck) &&
                (ev.taskStatus.equals (AppNotification.STATUS_COMPLETED) ||
                 ev.taskStatus.equals (AppNotification.STATUS_FAILED)))
            {
                    try
                    {
                            _soapClient.removeNotificationListener (on, this);
                    }
            catch (Throwable th)
                    {
                        System.out.println ("Error removing listener: " + th);
                    }
                    System.exit (0);
        }
    }
}
```

Adding to, updating, or deleting part of an application through programming

You can add to, update, or delete part of an existing application through the administrative console, the wsadmin tool, or programming. This example changes part of an application through programming. You can use this example whether you add to, update, or delete part of an existing application. Multiple changes to an application can be packaged in a single .zip file.

To learn about the structure of the .zip file, see updating applications through the administrative console.

This task assumes a basic familiarity with MBean programming. For information on MBean programming see MBean Java application programming interface (API) documentation.

Before you can add to, update, or delete part of an application on WebSphere Application Server, you must first install your application.

Perform the following tasks to add to, update, or delete part of an application through programming.

1. Connect to WebSphere Application Server.
2. Create the application management proxy.
3. Create the notification filter.
4. Add the listener.
5. Partially change the existing application.
6. Wait for some timeout so that the program does not end.
7. Listen to Java Management Extensions (JMX) notifications to understand completion of the operation.
8. When the update is done, remove the listener and quit.

After you successfully run the code, you have changed the application.

The following example shows how to add to, update, or delete part of an application based on the previous steps:

```
//Inputs:
//partialApp specifies the location of the partial application.
//appName specifies the name of the application.

String partialApp = "C:/apps/partial.zip";
String appName = "MyApp";

//Do a get of the administrative client to connect to
//WebSphere Application Server.

AdminClient client = ...;

//Create the application management proxy.
AppManagement proxy = AppManagementProxy. getJMXProxyForClient (client);

// Create the notification filter.
NotificationFilterSupport myFilter = new NotificationFilterSupport();
myFilter.enableType (NotificationConstants.TYPE_APPMANAGEMENT);
//Add the listener.
NotificationListener listener = new AListener(_soapClient, myFilter,
    "Install: " + appName, AppNotification.UPDATE);
//Partially change the existing application, MyApp.

Hashtable options = new Hashtable();
options.put (AppConstants.APPDEPL_LOCALE, Locale.getDefault());
options.put (AppConstants.APPUPDATE_CONTENTTYPE, AppConstants.
     APPUPDATE_CONTENT_PARTIALAPP);

proxy.updateApplication ( appName,
    null,
    partialApp,
    null,
    options,
    null);

// Wait for some timeout. The installation application programming interface (API) is
//  asynchronous and so returns immediately.
// If the program does not wait here, the program ends.
  Thread.sleep(300000); // Wait so that the program does not end.
        }
        catch (Exception e) {
            e.printStackTrace();
        }

    }

}
// Specify the Java Management Extensions (JMX) notification listener for JMX events.
class AListener implements NotificationListener
{
    AdminClient _soapClient;
    NotificationFilterSupport  myFilter;
    Object handback;
    ObjectName on;
    String eventTypeToCheck;

    public AListener(AdminClient cl, NotificationFilterSupport fl, Object h,
        String eType) throws Exception
    {
        _soapClient = cl;
        myFilter = fl;
        handback = h;
        eventTypeToCheck = eType;

        Iterator iter = _soapClient.queryNames (new ObjectName("WebSphere:
            type=AppManagement,*"), null).iterator();
        on = (ObjectName)iter.next();
        System.out.println ("ObjectName: " + on);
        _soapClient.addNotificationListener (on, this, myFilter, handback);
    }

    public void handleNotification (Notification notf, Object handback)
    {
            AppNotification ev = (AppNotification) notf.getUserData();
            System.out.println ("!! JMX event Recd: (handback obj= " +
                handback+ "): " + ev);


            //When the installation is done, remove the listener and quit

            if (ev.taskName.equals (eventTypeToCheck) &&
                (ev.taskStatus.equals (AppNotification.STATUS_COMPLETED) ||
                 ev.taskStatus.equals (AppNotification.STATUS_FAILED)))
            {
                    try
                    {
                            _soapClient.removeNotificationListener (on, this);
                    }
            catch (Throwable th)
                    {
                        System.out.println ("Error removing listener: " + th);
                    }
                    System.exit (0);
        }
    }
}
```

Preparing a module and adding it to an existing application through programming

You can add a module to an existing application through the administrative console, the wsadmin tool, or programming. Use this example to add a module through programming.

This task assumes a basic familiarity with MBean programming. For information on MBean programming see MBean Java application programming interface (API) documentation.

Before you can add a module to an application on WebSphere Application Server, you must install the application.

Perform the following tasks to add a module to an application through programming.

1. Create an application deployment controller instance to populate the module file with binding information.
2. Save the binding information in the module.
3. Get the installation options.
4. If the preparation phase (population of the EAR file) is not performed, the do the following actions:
   a. Create an options table to be passed to the updateApplication MBean API.
   b. Create a table for module to server relations and add the table to the options table.
5. Connect to WebSphere Application Server.
6. Create the application management proxy.
7. Create the notification filter.
8. Add the listener.
9. Add the module to the application.
10. Specify the target for the new module.
11. Wait for some timeout so that the program does not end.
12. Listen to Java Management Extensions (JMX) notifications to understand completion of the operation.
13. When the module addition is done, remove the listener and quit.

After you successfully run the code, the module is added to the application.

The following example shows how to add a module to an application based on the previous steps:

```
//Inputs:
//moduleName specifies the name of the module that you add to the application.
//moduleURI specifies a URI that gives the target location of the module
// archive contents on a file system. The URI provides the location of the new
// module after installation. The URI is relative to the application URL.
//uniquemoduleURI specfies the URI that gives the target location of the
// deployment descriptor file. The URI is relative to the application URL.
//target specifies the cell, node, and server on which the module is installed.

String moduleName = "C:/apps/foo,jar";
String moduleURI = "Increment.jar";
String uniquemoduleURI = "Increment.jar+META-INF/ejb-jar.xml";
String target = "WebSphere:cell=cellname,node=nodename,server=servername";

//Create an application deployment controller instance, AppDeploymentController,
//to populate the Java aArchive (JAR) file with binding information.
//The binding information is WebSphere Application Server-specific deployment information.

Hashtable preferences = new Hashtable();
preferences.put (AppConstants.APPDEPL_LOCALE, Locale.getDefault());
preferences.put (AppConstants.APPUPDATE_CONTENTTYPE, AppConstants.APPUPDATE_
    CONTENT_MODULEFILE);
AppDeploymentController controller = AppManagementFactory.readArchiveForUpdate(
    moduleName,
    moduleURI,
    AppConstants.APPUPDATE_ADD,
    preferences,
    null);
```

If the module that you add to the application lacks any bindings, add the bindings so that the module addition works. Collect and add the bindings by using the public APIs provided with WebSphere Application Server. Refer to Java documentation for the com.ibm.websphere.management.application.client.AppDeploymentController instance to learn more about how to collect and populate tasks with WebSphere Application Server specific-binding information.

```
//After you collect all the binding information, save it in the module.
controller.saveAndClose();

//Get the installation options.
Hashtable options = controller. getAppDeploymentSavedResults();

//Connect the administrative client, AdminClient, to WebSphere Application Server.
    AdminClient client = ...;

//Create the application management proxy.
AppManagement proxy = AppManagementProxy. getJMXProxyForClient (client);

//Update the existing application, MyApp, by adding the module.
String appName = "MyApp";

options.put (AppConstants.APPUPDATE_CONTENTTYPE,
   AppConstants. APPUPDATE_CONTENT_MODULEFILE);

//Create the notification filter.
  NotificationFilterSupport myFilter = new NotificationFilterSupport();
  myFilter.enableType (NotificationConstants.TYPE_APPMANAGEMENT);
  //Add the listener.
  NotificationListener listener = new AListener(_soapClient, myFilter,
      "Install: " + appName, AppNotification.UPDATE);



//Specify the target for the new module.
Hashtable mod2svr = new Hashtable();
options.put (AppConstants.APPDEPL_MODULE_TO_SERVER, mod2svr);
mod2svr.put (uniquemoduleURI, target);
proxy.updateApplication ( appName,
    moduleURI,
    moduleName,
    AppConstants.APPUPDATE_ADD,
    options,
    null);

// Wait for some timeout. The installation application programming interface (API) is
//  asynchronous and so returns immediately.
// If the program does not wait here, the program ends.
  Thread.sleep(300000); // Wait so that the program does not end.
        }
        catch (Exception e) {
            e.printStackTrace();
        }

    }

}
// Specify the Java Management Extensions (JMX) notification listener for JMX events.
class AListener implements NotificationListener
{
    AdminClient _soapClient;
    NotificationFilterSupport  myFilter;
    Object handback;
    ObjectName on;
    String eventTypeToCheck;

    public AListener(AdminClient cl, NotificationFilterSupport fl, Object h,
        String eType) throws Exception
    {
        _soapClient = cl;
        myFilter = fl;
        handback = h;
        eventTypeToCheck = eType;

        Iterator iter = _soapClient.queryNames (new ObjectName("WebSphere:
            type=AppManagement,*"), null).iterator();
        on = (ObjectName)iter.next();
        System.out.println ("ObjectName: " + on);
        _soapClient.addNotificationListener (on, this, myFilter, handback);
    }

    public void handleNotification (Notification notf, Object handback)
    {
            AppNotification ev = (AppNotification) notf.getUserData();
            System.out.println ("!! JMX event Recd: (handback obj= " +
                handback+ "): " + ev);


            //When the installation is done, remove the listener and quit

            if (ev.taskName.equals (eventTypeToCheck) &&
                (ev.taskStatus.equals (AppNotification.STATUS_COMPLETED) ||
                 ev.taskStatus.equals (AppNotification.STATUS_FAILED)))
            {
                    try
                    {
                            _soapClient.removeNotificationListener (on, this);
                    }
            catch (Throwable th)
                    {
                        System.out.println ("Error removing listener: " + th);
                    }
                    System.exit (0);
        }
    }
}
```

Preparing and updating a module through programming

You can update a module for an existing application through the administrative console, the wsadmin tool, or programming. When you update a module, you replace the existing module with a new version. Use this example to update a module through programming.

This task assumes a basic familiarity with MBean programming. For information on MBean programming see MBean Java application programming interface (API) documentation.

Before you can update a module on WebSphere Application Server, you must first install the application.

Perform the following tasks to update a module through programming.

1. Create an application deployment controller instance to populate the Java archive file with binding information.
2. Save the binding information in the module.
3. Get the installation options.
4. If the preparation phase (population of the EAR file) is not performed, the do the following actions:
   a. Create an options table to be passed to the updateApplication MBean API.
   b. Create a table for module to server relations and add the table to the options table.

5. Connect to WebSphere Application Server.
6. Create the application management proxy.
7. Create the notification filter.
8. Add the listener.
9. Replace the module in the application.
10. Specify the target for the new module.
11. Wait for some timeout so that the program does not end.
12. Listen to Java Management Extensions (JMX) notifications to understand completion of the operation.
13. When the module addition is done, remove the listener and quit.

After you successfully run the code, the existing module is replaced with the new one.

The following example shows how to add a module to an application based on the previous steps:

```
//Inputs:
//moduleName specifies the name of the module that you add to the application.
//moduleURI specifies a URI that gives the target location of the module
// archive contents on a file system. The URI provides the location of the new
// module after installation. The URI is relative to the application URL.
//uniquemoduleURI specfies the URI that gives the target location of the
// deployment descriptor file. The URI is relative to the application URL.
//target specifies the cell, node, and server on which the module is installed.
//appName specifies the name of the application to update.
String moduleName = "C:/apps/foo,jar";
String moduleURI = "Increment.jar";
String uniquemoduleURI = "Increment.jar+META-INF/ejb-jar.xml";
String target = "WebSphere:cell=cellname,node=nodename,server=servername";
String appName = "MyApp";

//Get the administrative client to connect to
//WebSphere Application Server.
AdminClient client = ...;
AppManagement proxy = AppManagementProxy. getJMXProxyForClient (client);

Vector tasks = proxy.getApplicationInfo (appName, new Hashtable(), null);

//Create an application deployment controller instance, AppDeploymentController,
//to populate the Java archive (JAR) file with binding information.
//The binding information is WebSphere Application Server-specific deployment information.

Hashtable preferences = new Hashtable();
preferences.put (AppConstants.APPDEPL_LOCALE, Locale.getDefault());
preferences.put (AppConstants.APPUPDATE_CONTENTTYPE, AppConstants.APPUPDATE_
    CONTENT_MODULEFILE);
AppDeploymentController controller = AppManagementFactory.readArchiveForUpdate(
    moduleName,
    moduleURI,
    AppConstants.APPUPDATE_UPDATE,
    preferences,
    tasks);
```

If the module that you update for the application lacks any bindings, add the bindings so that the module update works. Collect and add the bindings by using the public APIs that are provided with WebSphere Application Server. Refer to Java documentation for the AppDeploymentController instance to learn more about how to collect and populate tasks with WebSphere Application Server-specific binding information.

```
//After you collect all the binding information, save it in the module.
controller.saveAndClose();

//Create the notification filter.
  NotificationFilterSupport myFilter = new NotificationFilterSupport();
  myFilter.enableType (NotificationConstants.TYPE_APPMANAGEMENT);
  //Add the listener.
  NotificationListener listener = new AListener(_soapClient, myFilter, "Install:
      " + appName, AppNotification.UPDATE);


//Get the installation options.
Hashtable options = controller. getAppDeploymentSavedResults();

//Update the existing application by adding the module.

options.put (AppConstants.APPUPDATE_CONTENTTYPE,
   AppConstants. APPUPDATE_CONTENT_MODULEFILE);

//Specify the target for the new module
Hashtable mod2svr = new Hashtable();
options.put (AppConstants.APPDEPL_MODULE_TO_SERVER, mod2svr);
mod2svr.put (uniquemoduleURI, target);

proxy.updateApplication ( appName,
    moduleURI,
    moduleName,
    AppConstants.APPUPDATE_UPDATE,
    options,
    null);
// Wait; the installation application programming interface (API) is
//  asynchronous and so returns immediately.
// If the program does not wait here, the program ends.
  Thread.sleep(300000); // Wait so that the program does not end.
        }
        catch (Exception e) {
            e.printStackTrace();
        }

    }

}
// Specify the Java Management Extensions (JMX) notification listener for JMX events.
class AListener implements NotificationListener
{
    AdminClient _soapClient;
    NotificationFilterSupport  myFilter;
    Object handback;
    ObjectName on;
    String eventTypeToCheck;

    public AListener(AdminClient cl, NotificationFilterSupport fl, Object h,
         String eType) throws Exception
    {
        _soapClient = cl;
        myFilter = fl;
        handback = h;
        eventTypeToCheck = eType;

        Iterator iter = _soapClient.queryNames (new ObjectName("WebSphere:type=
             AppManagement,*"), null).iterator();
        on = (ObjectName)iter.next();
        System.out.println ("ObjectName: " + on);
        _soapClient.addNotificationListener (on, this, myFilter, handback);
    }

    public void handleNotification (Notification notf, Object handback)
    {
            AppNotification ev = (AppNotification) notf.getUserData();
            System.out.println ("!! JMX event Recd: (handback obj= " +
                 handback+ "): " + ev);
            //When the installation is done, remove the listener and quit

            if (ev.taskName.equals (eventTypeToCheck) &&
                (ev.taskStatus.equals (AppNotification.STATUS_COMPLETED) ||
                 ev.taskStatus.equals (AppNotification.STATUS_FAILED)))
            {
                    try
                    {
                            _soapClient.removeNotificationListener (on, this);
                    }
            catch (Throwable th)
                    {
                        System.out.println ("Error removing listener: " + th);
                    }
                    System.exit (0);
        }
    }
}
```

Deleting a module through programming

You can delete a module from an existing application through the administrative console, the wsadmin tool, or programming. Use this example to delete a module through programming.

This task assumes a basic familiarity with MBean programming. For information on MBean programming see MBean Java application programming interface (API) documentation.

Before you can delete a module from an application on WebSphere Application Server, you must first install the application.

Perform the following tasks to delete a module through programming.

1. Connect to WebSphere Application Server.
2. Create the application management proxy.
3. Create the notification filter for listening to events.
4. Add the listener.
5. Delete the module.
6. Wait for some timeout so that the program does not end.
7. Listen to Java Management Extensions (JMX) notifications to understand completion of the operation.
8. When the module is deleted, remove the listener and quit.

After you successfully run the code, the existing module is deleted from the application.

The following example shows how to delete a module from an application based on the previous steps:

```
//moduleURI specifies a URI that gives the target location of the module.
//appName specifies the name of the application to update.
String moduleURI = "Increment.jar";
String appName = "MyApp";

//Get the administrative client to connect to
//WebSphere Application Server.
AdminClient client = ...;

//Create the application management proxy.

AppManagement proxy = AppManagementProxy. getJMXProxyForClient (client);

//Create the notification filter.
  NotificationFilterSupport myFilter = new NotificationFilterSupport();
  myFilter.enableType (NotificationConstants.TYPE_APPMANAGEMENT);
  //Add the listener.
  NotificationListener listener = new AListener(_soapClient, myFilter, "Install: " + appName, AppNotification.UPDATE);
```

```
//Update the existing application, MyApp, by deleting the module.
Hashtable options = new Hashtable();
options.put (AppConstants.APPDEPL_LOCALE, Locale.getDefault());
options.put (AppConstants.APPUPDATE_CONTENTTYPE, AppConstants.APPUPDATE_CONTENT_MODULEFILE);

proxy.updateApplication ( appName,
    moduleURI,
    null,
    AppConstants.APPUPDATE_DELETE,
    options,
    null);

// Wait; the installation application programming interface (API) is
//  asynchronous and so returns immediately.
// If the program does not wait here, the program ends.
  Thread.sleep(300000); // Wait so that the program does not end.
        }
        catch (Exception e) {
            e.printStackTrace();
        }

    }

}
// Specify the Java Management Extensions (JMX) notification listener for JMX events.
class AListener implements NotificationListener
{
    AdminClient _soapClient;
    NotificationFilterSupport  myFilter;
    Object handback;
    ObjectName on;
    String eventTypeToCheck;

    public AListener(AdminClient cl, NotificationFilterSupport fl, Object h,
         String eType) throws Exception
    {
        _soapClient = cl;
        myFilter = fl;
        handback = h;
        eventTypeToCheck = eType;

        Iterator iter = _soapClient.queryNames (new ObjectName("WebSphere:
            type=AppManagement,*"), null).iterator();
        on = (ObjectName)iter.next();
        System.out.println ("ObjectName: " + on);
        _soapClient.addNotificationListener (on, this, myFilter, handback);
    }

    public void handleNotification (Notification notf, Object handback)
    {
            AppNotification ev = (AppNotification) notf.getUserData();
            System.out.println ("!! JMX event Recd: (handback obj= " +
                handback+ "): " + ev);


            //When the installation is done, remove the listener and quit

            if (ev.taskName.equals (eventTypeToCheck) &&
                (ev.taskStatus.equals (AppNotification.STATUS_COMPLETED) ||
                 ev.taskStatus.equals (AppNotification.STATUS_FAILED)))
            {
                    try
                    {
                            _soapClient.removeNotificationListener (on, this);
                    }
            catch (Throwable th)
```

```
                    {
                        System.out.println ("Error removing listener: " + th);
                    }
                    System.exit (0);
            }
        }
}
```

Adding a file through programming

You can add a file to an existing application through the administrative console, the wsadmin tool, or programming. This example describes how to add a file through programming.

This task assumes a basic familiarity with MBean programming. For information on MBean programming see MBean Java application programming interface (API) documentation.

Before you can add a file to an application on WebSphere Application Server, you must first install the application.

Perform the following tasks to add a file to an application through programming.

1. Connect to WebSphere Application Server.
2. Create the application management proxy.
3. Create the notification filter for listening to events.
4. Add the listener.
5. Add the file to the application.
6. Wait for some timeout so that the program does not end.
7. Listen to Java Management Extensions (JMX) notifications to understand completion of the operation.
8. When the file is added to the application, remove the listener and quit.

After you successfully run the code, the file is added to the application.

The following example shows how to add a file to an application based on the previous steps:

```
import java.lang.*;
import java.io.*;
import java.util.*;
import java.lang.reflect.*;
import com.ibm.websphere.management.application.*;
import com.ibm.websphere.management.application.client.*;
import com.ibm.websphere.management.*;

import javax.management.*;

public class FileAdd {

    public static void main (String [] args) {

        try {

// Get a connection to WebSphere Application Server.
  String host = "localhost";
  String port = "8880";
  String target = "WebSphere:cell=cellName,node=nodeName,server=server1";

  Properties config = new Properties();
  config.put (AdminClient.CONNECTOR_HOST,  host);
  config.put (AdminClient.CONNECTOR_PORT,  port);
  config.put (AdminClient.CONNECTOR_TYPE, AdminClient.CONNECTOR_TYPE_SOAP);
  System.out.println ("Config: " + config);
     AdminClient _soapClient = AdminClientFactory.createAdminClient(config);

 // Create the application management proxy, AppManagement.
```

```
  AppManagement proxy = AppManagementProxy. getJMXProxyForClient (_soapClient);

  String appName = "MyApp";
  String fileURI = "test.war/com/acme/abc.jsp";
  String fileContents = "C:/temp/abc.jsp";

  //Create the notification filter.
  NotificationFilterSupport myFilter = new NotificationFilterSupport();
  myFilter.enableType (NotificationConstants.TYPE_APPMANAGEMENT);

  //Add the listener.
  NotificationListener listener = new AListener(_soapClient, myFilter,
     "Install: " + appName, AppNotification.UPDATE);

  Hashtable options = new Hashtable();
  options.put (AppConstants.APPDEPL_LOCALE, Locale.getDefault());
  options.put (AppConstants.APPUPDATE_CONTENTTYPE, AppConstants.APPUPDATE_CONTENT_FILE);

  // Update the application
  proxy.updateApplication (   appName,
                 fileURI,
                 fileContents,
                 AppConstants.APPUPDATE_ADD,
                 options,
                 null);

// Wait; the installation Application Programming Interface (API) is
//  asynchronous and so returns immediately.
// If the program does not wait here, the program ends.
  Thread.sleep(90000); // Wait so that the program does not end.

        }
        catch (Exception e) {
            e.printStackTrace();
        }

    }

}


// Specify the Java Management Extensions (JMX) notification listener for JMX events.
class AListener implements NotificationListener
{
    AdminClient _soapClient;
    NotificationFilterSupport  myFilter;
    Object handback;
    ObjectName on;
    String eventTypeToCheck;

    public AListener(AdminClient cl, NotificationFilterSupport fl, Object h,
        String eType) throws Exception
    {
        _soapClient = cl;
        myFilter = fl;
        handback = h;
        eventTypeToCheck = eType;

        Iterator iter = _soapClient.queryNames (new ObjectName("WebSphere:
           type=AppManagement,*"), null).iterator();
        on = (ObjectName)iter.next();
        System.out.println ("ObjectName: " + on);
        _soapClient.addNotificationListener (on, this, myFilter, handback);
    }

    public void handleNotification (Notification notf, Object handback)
    {
            AppNotification ev = (AppNotification) notf.getUserData();
            System.out.println ("!! JMX event Recd: (handback obj= " +
                handback+ "): " + ev);


            //When the installation is done, remove the listener and quit

            if (ev.taskName.equals (eventTypeToCheck) &&
                (ev.taskStatus.equals (AppNotification.STATUS_COMPLETED) ||
                 ev.taskStatus.equals (AppNotification.STATUS_FAILED)))
            {
                    try
                    {
                            _soapClient.removeNotificationListener (on, this);
                    }
            catch (Throwable th)
                    {
                        System.out.println ("Error removing listener: " + th);
                    }
                    System.exit (0);
        }
    }
}
```

Updating a file through programming

You can update a file for an existing application through the administrative console, the wsadmin tool, or programming. This example describes how to update a file through programming.

This task assumes a basic familiarity with MBean programming. For information on MBean programming see MBean Java application programming interface (API) documentation.

Before you can update a file for an application on WebSphere Application Server, you must first install the application.

Perform the following tasks to update a file through programming.

1. Connect to WebSphere Application Server.
2. Create the application management proxy.
3. Create the notification filter for listening to events.
4. Add the listener.
5. Update the file in the application.
6. Wait for some timeout so that the program does not end.
7. Listen to Java Management Extensions (JMX) notifications to understand completion of the operation.
8. When the installation is done, remove the listener and quit.

After you successfully run the code, the file is updated for the application.

The following example shows how to add a file to an application based on the previous steps:

```
//Inputs:
//fileContents specifies the name of the file that you add to the application.
//appName specifies the name of the application.
//fileURI specifies a URI that gives the target location of the file. The URI
// provides the location of the new module after installation. The URI is
// relative to the application URL.

String fileContents = "C:/apps/test.jsp";
String appName = "MyApp";
String fileURI = "SomeWebMod.war/com/foo/abc.jsp";

//Get the administrative client to connect to
//WebSphere Application Server.
```

```
AdminClient client = ...;

//Create the application management proxy.
AppManagement proxy = AppManagementProxy. getJMXProxyForClient (client);

//Create the notification filter.
  NotificationFilterSupport myFilter = new NotificationFilterSupport();
  myFilter.enableType (NotificationConstants.TYPE_APPMANAGEMENT);
  //Add the listener.
  NotificationListener listener = new AListener(_soapClient, myFilter,
      "Install: " + appName, AppNotification.UPDATE);

Hashtable options = new Hashtable();
options.put (AppConstants.APPDEPL_LOCALE, Locale.getDefault());
options.put (AppConstants.APPUPDATE_CONTENTTYPE, AppConstants.APPUPDATE_CONTENT_FILE);

proxy.updateApplication ( appName,
    fileURI,
    fileContents,
    AppConstants.APPUPDATE_UPDATE,
    options,
    null);

// Wait; the installation application programming interface (API) is
//  asynchronous and so returns immediately.
// If the program does not wait here, the program ends.
  Thread.sleep(300000); // Wait so that the program does not end.
        }
        catch (Exception e) {
            e.printStackTrace();
        }

    }

}
// Specify the Java Management Extensions (JMX) notification listener for JMX events.
class AListener implements NotificationListener
{
    AdminClient _soapClient;
    NotificationFilterSupport  myFilter;
    Object handback;
    ObjectName on;
    String eventTypeToCheck;

    public AListener(AdminClient cl, NotificationFilterSupport fl, Object h,
         String eType) throws Exception
    {
        _soapClient = cl;
        myFilter = fl;
        handback = h;
        eventTypeToCheck = eType;

        Iterator iter = _soapClient.queryNames (new ObjectName("WebSphere:
            type=AppManagement,*"), null).iterator();
        on = (ObjectName)iter.next();
        System.out.println ("ObjectName: " + on);
        _soapClient.addNotificationListener (on, this, myFilter, handback);
    }

    public void handleNotification (Notification notf, Object handback)
    {
            AppNotification ev = (AppNotification) notf.getUserData();
            System.out.println ("!! JMX event Recd: (handback obj= " +
                handback+ "): " + ev);


            //When the installation is done, remove the listener and quit.
            if (ev.taskName.equals (eventTypeToCheck) &&
                (ev.taskStatus.equals (AppNotification.STATUS_COMPLETED) ||
                 ev.taskStatus.equals (AppNotification.STATUS_FAILED)))
            {
                    try
                    {
                            _soapClient.removeNotificationListener (on, this);
                    }
            catch (Throwable th)
                    {
                        System.out.println ("Error removing listener: " + th);
                    }
                    System.exit (0);
        }
    }
}
```

Deleting a file through programming

You can delete a file from an existing application through the administrative console, the wsadmin tool, or programming. Use this example to delete a file through programming.

This task assumes a basic familiarity with MBean programming. For information on MBean programming see MBean Java application programming interface (API) documentation.

Before you can delete a file from an application on WebSphere Application Server, you must first install the application.

Perform the following tasks to delete a file through programming.

1. Connect to WebSphere Application Server.
2. Create the application management proxy.
3. Create the notification filter for listening to events.
4. Add the listener.
5. Delete the file from the application.
6. Wait for some timeout so that the program does not end.
7. Listen to Java Management Extensions (JMX) notifications to understand completion of the operation.
8. When the file is deleted from the application, remove the listener and quit.

After you successfully run the code, the file is deleted from the application.

The following example shows how to delete a file based on the previous steps:

```
//Inputs:
//fileURI specifies a URI that gives the target location of the file. The URI
// provides the location of the new module after installation. The URI is
// relative to the application URL.
//appName specifies the name of the application.

String fileURI = "Increment.jar/com/acme/Foo.class";
String appName = "MyApp";

//Get the administrative client to connect to
//WebSphere Application Server.
AdminClient client = ...;

//Create the application management proxy.
AppManagement proxy = AppManagementProxy. getJMXProxyForClient (client);

//Create the notification filter.
  NotificationFilterSupport myFilter = new NotificationFilterSupport();
  myFilter.enableType (NotificationConstants.TYPE_APPMANAGEMENT);
  //Add the listener.
  NotificationListener listener = new AListener(_soapClient, myFilter,
      "Install: " + appName, AppNotification.UPDATE);

//Update the existing application, MyApp, by deleting the file.
Hashtable options = new Hashtable();
options.put (AppConstants.APPDEPL_LOCALE, Locale.getDefault());
options.put (AppConstants.APPUPDATE_CONTENTTYPE, AppConstants.APPUPDATE_CONTENT_FILE);

proxy.updateApplication ( appName,
    fileURI,
    null,
    AppConstants.APPUPDATE_DELETE,
    options,
    null);

// Wait for some timeout. The installation Application Programming Interface (API) is
//  asynchronous and so returns immediately.
// If the program does not wait here, the program ends.
  Thread.sleep(300000); // Wait so that the program does not end.
        }
        catch (Exception e) {
            e.printStackTrace();
        }

    }

}
// Specify the Java Management Extensions (JMX) notification listener for JMX events.
class AListener implements NotificationListener
{
    AdminClient _soapClient;
    NotificationFilterSupport  myFilter;
    Object handback;
    ObjectName on;
    String eventTypeToCheck;

    public AListener(AdminClient cl, NotificationFilterSupport fl, Object h,
         String eType) throws Exception
    {
        _soapClient = cl;
        myFilter = fl;
        handback = h;
        eventTypeToCheck = eType;

        Iterator iter = _soapClient.queryNames (new ObjectName("WebSphere:
            type=AppManagement,*"), null).iterator();
        on = (ObjectName)iter.next();
        System.out.println ("ObjectName: " + on);
        _soapClient.addNotificationListener (on, this, myFilter, handback);
    }

    public void handleNotification (Notification notf, Object handback)
    {
            AppNotification ev = (AppNotification) notf.getUserData();
            System.out.println ("!! JMX event Recd: (handback obj= " +
                 handback+ "): " + ev);


            //Once the installation is done, remove the listener and quit

            if (ev.taskName.equals (eventTypeToCheck) &&
                (ev.taskStatus.equals (AppNotification.STATUS_COMPLETED) ||
                 ev.taskStatus.equals (AppNotification.STATUS_FAILED)))
            {
                    try
                    {
```

```
                    _soapClient.removeNotificationListener (on, this);
                }
            catch (Throwable th)
                {
                    System.out.println ("Error removing listener: " + th);
                }
                System.exit (0);
        }
    }
}
```

Using command line tools

There are several command line tools that you can use to start, stop, and monitor WebSphere server processes and nodes. These tools only work on local servers and nodes. They cannot operate on a remote server or node. To administer a remote server, you can use the wsadmin scripting program connected to the deployment manager for the cell in which the target server or node is configured. See Deploying and managing using scripting for more information about using the wsadmin scripting program. You can also use the V5 administrative console which runs in the deployment manager for the cell. For more information about using the administrative console, see Deploying and managing with the GUI.

All command line tools function relative to a particular profile. If you run a command from a *install_root*/WebSphere/AppServer/bin directory, the command will run within the default profile. If you want to specify a different profile, perform one of the following:

- Specify the -profileName option. The profile that you specify with this option will be used instead of the default profile. For example:
  1. Change to the *install_root*/WebSphere/AppServer/bin directory.
  2. Type the following command: startServer *server1* -profileName *AppServerProfile*

  In this example, the command will function inside the *AppServerProfile* profile.
- Run the command from the bin directory of a specific profile. For example:
  1. Change to the *install_root*/WebSphere/AppServer/profiles/*MyProfile*/bin directory.
  2. Type the following command: startServer *server1*

  In this example, the command will function inside the *MyProfile* profile.

For more information about using profiles, including how to obtain a list of profiles, see the wasprofile command article.

To use the command line tools, perform the following steps:

1. Open a system command prompt.
2. Change to the bin directory.
3. Run the command.

The command runs the requested function and displays the results on the screen. Refer to the command log file for additional information. When you use the -trace option for the command, the additional trace data is captured in the command log file. The directory location for the log files is under the default system log root directory, except for commands related to a specific server instance, in which case the log directory for that server is used. You can override the default location for the command log file using the -logfile option for the command.

Example: Security and the command line tools

If you want to enable WebSphere Application Server security, you need to provide the command line tools with authentication information. Without authentication information, the command line tools receive an `AccessDenied` exception when you attempt to use them with security enabled. There are multiple ways to provide authentication data:

- Most command line tools support a -username and -password option for providing basic authentication data. Specify the user ID and password for an administrative user. For example, you can use a member of the administrative console users with operator or administrator privileges, or the administrative user ID configured in the user registry. The following example demonstrates the **stopNode** command, which specifies command line parameters:

  ```
  stopNode -username adminuser -password adminpw
  ```
- You can place the authentication data in a properties file that the command line tools read. The default file for this data is the `sas.client.props` file in the `properties` directory for the WebSphere Application Server.

startServer command

The **startServer** command reads the configuration file for the specified application server and starts the server. Depending on the options you specify, you can launch a new Java virtual machine (JVM) API to run the server process, or write the launch command data to a file. For more information about where to run this command, see the Using command tools article.

If you are using the Windows platform and the you have the application server running as a Windows service, the **startServer** command will start the associated Windows service and it will be responsible for starting the application server.

Syntax

The command syntax is as follows:

```
startServer <server> [options]
```

where `server` is the name of the application server you want to start. This argument is required.

Parameters

The following options are available for the **startServer** command:

-nowait
Tells the **startServer** command not to wait for successful initialization of the launched server process.

-quiet
Suppresses the progress information that the **startServer** command prints in normal mode.

-logfile <fileName>
Specifies the location of the log file to which information is written.

-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. The default for this option is the default profile.

-replacelog
Replaces the log file instead of appending to the current log.

-trace
Generates trace information to the log file for debugging purposes.

-timeout <seconds>
Specifies the waiting time before server initialization times out and returns an error.

-statusport <portNumber>
Specifies that an administrator can set the port number for server status callback.

-script [<script fileName>] -background
Generates a launch script with the **startServer** command instead of launching the server process directly. The launch script name is an optional argument. If you do not supply the launch script name, the default script file name is `start_<server>` based on the *<server>* name passed as the first argument to the **startServer** command. The `-background` parameter is an optional parameter that specifies that the generated script will run in the background when you execute it.

-J <java_option>
Specifies options to pass through to the Java interpreter.

-help
Prints a usage statement.

-? Prints a usage statement.

Usage scenario

The following examples demonstrate correct syntax:

```
startServer server1

startServer server1 -script (produces the start_server1.sh or .bat files)

startServer server1 -trace (produces the startserver.log file)
```

stopServer command

The **stopServer** command reads the configuration file for the specified server process. This command sends a Java Management Extensions (JMX) command to the server telling it to shut down. By default, the **stopServer** command does not return control to the command line until the server completes the shut down process. There is a -nowait option to return immediately, as well as other options to control the behavior of the **stopServer** command. For more information about where to run this command, see the Using command tools article.

If you are using the Windows platform and the you have the application server running as a Windows service, the **stopServer** command will start the associated Windows service and it will be responsible for starting the application server.

Syntax

The command syntax is as follows:

```
stopServer <server> [options]
```

where *server* is the name of the configuration directory of the server you want to stop. This argument is required.

Parameters

The following options are available for the **stopServer** command:

-nowait
Tells the **stopServer** command not to wait for successful shutdown of the server process.

-quiet
Suppresses the progress information that the **stopServer** command prints in normal mode.

-logfile <fileName>
Specifies the location of the log file to which information is written.

-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. The default for this option is the default profile.

-replacelog
Replaces the log file instead of appending to the current log.

-trace
Generates trace information into a file for debugging purposes.

-timeout <seconds>
Specifies the time to wait for server shutdown before timing out and returning an error.

-statusport <portNumber>
Supports an administrator in setting the port number for server status callback.

-conntype <type>
Specifies the Java Management Extensions (JMX) connector type to use for connecting to the deployment manager. Valid types are Simple Object Access Protocol (SOAP), or Remote Method Invocation (RMI).

-port <portNumber>
Specifies the server Java Management Extensions (JMX) port to use explicitly, so that you can avoid reading the configuration files to obtain the information.

-username <name>
Specifies the user name for authentication if security is enabled in the server. Acts the same as the -user option.

-user <name>
Specifies the user name for authentication if security is enabled in the server. Acts the same as the -username option.

-password <password>
Specifies the password for authentication if security is enabled in the server.

Note: If you are running in a secure environment but have not provided a user ID and password, you will receive the following error message:

```
ADMN0022E: Access denied for the stop operation on Server MBean due
to insufficient or empty credentials.
```

To solve this problem, provide the user ID and password information.

-help
Prints a usage statement.

-? Prints a usage statement.

Usage scenario

The following examples demonstrate correct syntax:

```
stopServer server1

stopServer server1 -nowait

stopServer server1 -trace (produces the stopserver.log file)
```

startManager command

The **startManager** command reads the configuration file for the Network Deployment manager process and constructs a **launch** command. Depending on the options you specify, the **startManager** command launches a new Java virtual machine (JVM) API to run the manager process, or writes the **launch** command data to a file. You must run this command from the *install_root*/WebSphere/AppServer/profiles/standalone/bin directory of a Network Deployment installation.

If you are using the Windows platform and the you have the deployment manager running as a Windows service, the **startManager** command will start the associated Windows service and it will be responsible for starting the deployment manager.

Syntax

The command syntax is as follows:

```
startManager [options]
```

Parameters

The following options are available for the **startManager** command:

-nowait
Tells the **startManager** command not to wait for successful initialization of the deployment manager process.

-quiet
Suppresses the progress information that the **startManager** command prints in normal mode.

-logfile <fileName>
Specifies the location of the log file to which information gets written.

-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. The default for this option is the default profile.

-replacelog
Replaces the log file instead of appending to the current log.

-trace
Generates trace information into a file using the **startManager** command for debugging purposes.

-timeout <seconds>
Specifies the waiting time before deployment manager initialization times out and returns an error.

-statusport <portNumber>
Specifies that an administrator can set the port number for deployment manager status callback.

-script [<script fileName>] -background
Generates a launch script with the **startManager** command instead of launching the deployment manager process directly. The launch script name is an optional argument. If you do not provide the launch script name, the default script file name is `<start_dmgr>`. The `-background` parameter is an optional parameter that specifies that the generated script will run in the background when you execute it.

-J-<java_option>
Specifies options to pass through to the Java interpreter.

-help
Prints a usage statement.

-? Prints a usage statement.

Usage scenario

The following examples demonstrate correct syntax:

```
startManager

startManager -script (produces the start_dmgr.sh or .bat file)

startManager -trace (produces the startmanager.log file)
```

stopManager command

The **stopManager** command reads the configuration file for the Network Deployment manager process. It sends a Java Management Extensions (JMX) command to the manager telling it to shut down. By default, the **stopManager** command waits for the manager to complete the shutdown process before it returns control to the command line. There is a -nowait option to return immediately, as well as other options to control the behavior of the **stopManager** command. For more information about where to run this command, see the Using command tools article.

If you are using the Windows platform and the you have the deployment manager running as a Windows service, the **stopManager** command will start the associated Windows service and it will be responsible for starting the deployment manager.

Syntax

The command syntax is as follows:

```
stopManager [options]
```

Parameters

The following options are available for the **stopManager** command:

-nowait
Tells the **stopManager** command not to wait for successful shutdown of the deployment manager process.

-quiet
Suppresses the progress information that the **stopManager** command prints in normal mode.

-logfile <fileName>
Specifies the location of the log file to which information is written.

-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. The default for this option is the default profile.

-replacelog
Replaces the log file instead of appending to the current log.

-trace
Generates trace information to a file for debugging purposes.

-timeout <seconds>
Specifies the waiting time for the manager to complete shutdown before timing out and returning an error.

-statusport <portNumber>
Specifies that an administrator can set the port number for server status callback.

-conntype <type>
Specifies the Java Management Extensions (JMX) connector type to use for connecting to the deployment manager. Valid types are Simple Object Access Protocol (SOAP) or Remote Method Invocation (RMI).

-port <portNumber>
Specifies the deployment manager JMX port to use explicitly, so that you can avoid reading the configuration files to obtain information.

-username <name>
Specifies the user name for authentication if security is enabled in the deployment manager. Acts the same as the -user option.

-user <name>
Specifies the user name for authentication if security is enabled in the deployment manager. Acts the same as the -username option.

-password <password>
Specifies the password for authentication if security is enabled in the deployment manager.

Note: If you are running in a secure environment but have not provided a user ID and password, you receive the following error message:

```
ADMN0022E: Access denied for the stop operation on Server MBean due
to insufficient or empty credentials.
```

To solve this problem, provide the user ID and password information.

-help
Prints a usage statement.

-? Prints a usage statement.

Usage scenario

The following examples demonstrate correct syntax:

```
stopManager

stopManager -nowait

stopManager -trace (produces the stopmanager.log file)
```

startNode command

The **startNode** command reads the configuration file for the node agent process and constructs a **launch** command. Depending on the options that you specify, the **startNode** command creates a new Java virtual machine (JVM) API to run the agent process, or writes the launch command data to a file. For more information about where to run this command, see the Using command tools article.

If you are using the Windows platform and the you have the node agent running as a Windows service, the **startNode** command will start the associated Windows service and it will be responsible for starting the node agent.

Syntax

The command syntax is as follows:

```
startNode [options]
```

Parameters

The following options are available for the **startNode** command:

-nowait
Tells the **startNode** command not to wait for successful initialization of the node agent process.

-quiet
Suppresses the progress information that the **startNode** command prints in normal mode.

-logfile <fileName>
Specifies the location of the log file to which information gets written.

-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. The default for this option is the default profile.

-replacelog
Replaces the log file instead of appending to the current log.

-trace
Generates trace information into a file for debugging purposes.

-timeout <seconds>
Specifies the waiting time before node agent initialization times out and returns an error.

-statusport <portNumber>
Specifies that an administrator can set the port number for node agent status callback.

-script [<script fileName>] -background
Generates a launch script with the **startNode** command instead of launching the node agent process directly. The launch script name is an optional argument. If you do not provide the launch script name, the default script file name is `start_<nodeName>`, based on the name of the node. The `-background` parameter is an optional parameter that specifies that the generated script will run in the background when you execute it.

-J-<java_option>
Specifies options to pass through to the Java interpreter.

-help
Prints a usage statement.

-? Prints a usage statement.

Usage scenario

The following examples demonstrate correct syntax:

```
startNode

startNode -script (produces the start_node.bat or .sh file)

startNode -trace (produces the startnode.log file)
```

stopNode command

The **stopNode** command reads the configuration file for the Network Deployment node agent process and sends a Java Management Extensions (JMX) command telling the node agent to shut down. By default, the **stopNode** command waits for the node agent to complete shutdown before it returns control to the command line. There is a -nowait option to return immediately, as well as other options to control the behavior of the **stopNode** command. For more information about where to run this command, see the Using command tools article.

If you are using the Windows platform and the you have the node agent running as a Windows service, the **stopNode** command will start the associated Windows service and it will be responsible for starting the node agent.

Syntax

The command syntax is as follows:

```
stopNode [options]
```

Parameters

The following options are available for the **stopNode** command:

-nowait
Tells the **stopNode** command not to wait for successful shutdown of the node agent process.

-quiet
Suppresses the progress information that the **stopNode** command prints in normal mode.

-logfile <fileNname>
Specifies the location of the log file to which information gets written.

-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. The default for this option is the default profile.

-replacelog
Replaces the log file instead of appending to the current log.

-trace
Generates trace information into a file for debugging purposes.

-timeout <seconds>
Specifies the waiting time for the agent to shut down before timing out and returning an error.

-statusport <portNumber>
Specifies that an administrator can set the port number for server status callback.

-stopservers
Stops all application servers on the node before stopping the node agent.

-conntype <type>
Specifies the Java Management Extensions (JMX) connector type to use for connecting to the deployment manager. Valid types are Simple Object Access Protocol (SOAP) or Remote Method Invocation (RMI).

-port <portNumber>
Specifies the node agent JMX port to use explicitly, so that you can avoid reading configuration files to obtain the information.

-username <name>
Specifies the user name for authentication if security is enabled in the node agent. Acts the same as the -user option.

-user <name>
Specifies the user name for authentication if security is enabled in the node agent. Acts the same as the -username option.

-password <password>
Specifies the password for authentication if security is enabled in the node agent.

Note: If you are running in a secure environment but have not provided a user ID and password, you receive the following error message:

```
ADMN0022E: Access denied for the stop operation on Server MBean due
to insufficient or empty credentials.
```

To solve this problem, provide the user ID and password information.

-help
Prints a usage statement.

Note: When requesting help for the usage statement for the **stopNode** command, a reference to the **stopServer** command displays. All of the options displayed for this usage statement apply to the **stopNode** command.

-? Prints a usage statement.

Note: When requesting help for the usage statement for the **stopNode** command, a reference to the **stopServer** command displays. All of the options displayed for this usage statement apply to the **stopNode** command.

Usage scenario

The following examples demonstrate correct syntax:

```
stopNode

stopNode -nowait

stopNode -trace (produces the stopnode.log file)
```

addNode command

The **addNode** command incorporates a WebSphere Application Server installation into a cell. For more information about where to run this command, see the Using command tools article. Depending on the size and location of the new node you incorporate into the cell, this command can take a few minutes to complete.

The node agent server is automatically started as part of the **addNode** command unless you specify the `-noagent` option. If you recycle the system that hosts an application server node, and did not set up the node agent to act as an operating system daemon, you must issue a **startNode** command to start the node agent before starting any application servers.

The following items are new in V6:

- Ports generated for the node agent are unique for all the profiles in the installation. For development purposes, you can create multiple profiles on the same installation and add them to one or more cells without worrying about ports conflicts.
- If you want to specify the ports that the node agent uses, specify it is a file with the file name passed with the -portprops option. The format of the file is key=value pairs, one on each line, with the key being the same as the port name in the serverindex.xml file.
- If you want to use a number of sequential ports, the -startingport option works the same as it does in V5.x. This means that port conflicts with other profiles will not be detected.

Syntax

The command syntax is as follows:

```
addNode dmgr_host [dmgr_port] [-conntype type] [-includeapps]
[-startingport portnumber] [-portprops qualified_filename]
[-nodeagentshortname name] [-nodegroupname name] [-includebuses name]
[-registerservice] [-servicename name] [-servicepassword password]
[-coregroupname name] [-noagent] [-statusport port] [-quiet] [-nowait]
[-logfile filename] [-replacelog] [-trace] [-username uid]
[-password pwd] [-help]
```

The dmgr_host argument is required. All of the other arguments are optional. The default port number is 8879 for the default Simple Object Access Protocol (SOAP) port of the deployment manager. SOAP is the default Java Management Extensions (JMX) connector type for the command. If you have multiple WebSphere Application Server installations or multiple profiles, the SOAP port may be different than 8879. Examine the deployment manager SystemOut.log to see the current ports in use.

Parameters

The following options are available for the **addNode** command:

-conntype <type>
Specifies the JMX connector type to use for connecting to the deployment manager. Valid types are SOAP or RMI, which stands for Remote Method Invocation.

-includeapps
By default the **addNode** command does not carry over applications from the stand-alone servers on the new node to the cell. In general, you should install applications using the deployment manager. The -includeapps option tells the **addNode** command to carry over the applications from a node. If the application already exists in the cell, a warning is printed and the application does not install in the cell.

The applications will be mapped to the server that you federated using the **addNode** command. When the **addNode** command operation completes, the applications will run on that server when the server is started. Since these applications are part of the network deployment cell, you can map them to other servers and clusters in the cell using the administrative console. See the Mapping modules to servers article for more information.

By default, during application installation, application binaries are extracted in the *install_root*/installedApps/cellName directory. After the **addNode** command, the cell name of the configuration on the node that you added changes from the base cell name to the deployment manager cell name. The application binaries are located where they were before the **addNode** command ran, for example, *install_root*/installedApps/old_cellName.

If the application was installed by explicitly specifying the location for binaries as the following example:

```
${INSTALL_ROOT}/${CELL}
```

where the variable ${CELL}, specifies the current cell name, then when the **addNode** command runs, the binaries are moved to the following directory:

```
${INSTALL_ROOT}/currentCellName
```

Federating the node to a cell using the **addNode** command does not merge any cell level configuration, including virtual host information. If the virtual host and aliases for the new cell do not match WebSphere Application Server, you cannot access the applications running on the servers. You have to manually add all the virtual host and host aliases to the new cell, using the administrative console running on the deployment manager.

-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. The default for this option is the default profile.

-user <name> or -username <name>
Specifies the user name for authentication if security is enabled. Acts the same as the -user option. The user name that you choose must be a pre-existing user name.

-nowait
Tells the **addNode** command not to wait for successful initialization of the launched node agent process.

-quiet
Suppresses the progress information that the **addNode** command prints in normal mode.

-logfile <filename>
Specifies the location of the log file to which information gets written. By default, the log file is called addNode.log and is created in the `logs` directory of the profile for the node being added.

-trace
Generates additional trace information in the log file for debugging purposes.

-replacelog
Replaces the log file instead of appending to the current log. By default, the **addNode** command appends to the existing trace file. This option causes the **addNode** command to overwrite the trace file.

-noagent
Tells the **addNode** command not to launch the node agent process for the new node.

-password <password>
Specifies the password for authentication if security is enabled. The password that you choose must be one that is associated with a pre-existing user name.

-startingport <portNumber>
Supports the specification of a port number to use as the base port number for all node agent ports created during the **addNode** command. With this support you can control which ports are defined for these servers, rather than using the default port values. The starting port number is incremented to calculate the port number for every node agent port configured during the **addNode** command.

-registerservice
(Windows only) Registers the node agent as a Windows service.

-servicename <user>
(Windows only) Use the given user name as the Windows service user.

-servicepassword <password>
(Windows password) Use the given password as the Windows service password.

-portprops <filename>
Passes the name of the file that contains key-value pairs of explicit ports that you want the new node agent to use. For example, to set your SOAP and RMI ports to 3000 and 3001, create a file with the following two lines and pass it as the parameter:

```
SOAP_CONNECTOR_ADDRESS=3000
BOOTSTRAP_ADDRESS=3001
```

-coregroupname <name>
The name of the core group in which to add this node. If you do not specify this option, the node will be added to the DefaultCoreGroup.

-nodegroupname <name>
The name of the node group in which to add this node. If you do not specify, the node is added to the DefaultNodeGroup.

-includebuses
Copies the buses from the node to be federated to the cell.

-nodeagentshortname <name>
The shortname to use for the new node agent.

-help
Prints a usage statement.

-? Prints a usage statement.

Usage scenario

The following examples demonstrate correct syntax:

```
addNode testhost 8879 (adds an Application Server to the deployment manager)

addNode deploymgr 8879 -trace (produces the addNode.log file)

addNode host25 8879 -nowait (does not wait for a node agent process)
```

where 8879 is the default port.

Best practices for adding nodes using command line tools

Use the **addNode** command to add a standalone node into a cell. The **addNode** command does the following:

- Copies the base WebSphere Application Server cell configuration to a new cell structure. This new cell structure matches the structure of deployment manager.
- Creates a new node agent definition for the node that the cell incorporates.
- Sends commands to the deployment manager to add the documents from the new node to the cell repository.
- Performs the first configuration synchronization for the new node, and verifies that this node is synchronized with the cell.
- Launches the node agent process for the new node.
- Updates the setupCmdLine.bat or setupCmdline.sh files and the wsadmin.properties file to point to the new cell.
- After federating the node, the **addNode** command backs up the `plugin-cfg.xml` file from the `<install_root>/config/cells` directory to the `config/backup/base/cells` directory. The **addNode** command regenerates a new `plugin-cfg.xml` file at the Deployment Manger and the nodeSync operation copies the files to the node level.

  For information about port numbers, see port number settings in WebSphere Application Server versions.

Tips for using the **addNode** command:

- Do not put WebSphere Application Server Jar files on the generic `CLASSPATH` variable (default class path) for the overall system.
- **Unix/Linux users:** Some Unix or Linux systems create an association between the host name of the machine and the loopback address -- 127.0.0.1 (Red Hat installations do this by default). In addition, the `/etc/nsswitch.conf` file is set up to use the `/etc/hosts` path before trying to look up the server using a name server. This setup can cause failures when trying to add or administrate nodes when the deployment manager or application server is running on the Red Hat system or an Unix/Linux system with the same setup.

  If your deployment manager or your application server run on the Red Hat system, or an Unix/Linux system with the same setup, perform the following operations to ensure that you can successfully add and administer nodes:
  - Remove the 127.0.0.1 mapping to the local host in the `/etc/hosts` path.
  - Edit the `/etc/nsswitch.conf` file so that the hosts line reads:

    ```
    hosts:  dns files
    ```
- By default, applications that are installed on the node will not copy to the cell. If you install an application after using the **addNode** command, the application will install on the cell. By specifying the `-includeapps` option, you force the **addNode** command to copy applications from the node to the cell. Applications with duplicate names will not copy to the cell.
- Cell-level documents are not merged. Any changes that you make to the standalone cell-level documents before using the **addNode** command must be repeated on the new cell. For example, virtual hosts.

serverStatus command

Use the **serverStatus** command to obtain the status of one or all of the servers configured on a node. For more information about where to run this command, see the Using command tools article.

Syntax

The command syntax is as follows:

```
serverStatus <server>|-all [options]
```

The first argument is required. The argument is either the name of the server for which status is desired, or the -all keyword which requests status for all servers defined on the node.

Parameters

The following options are available for the **serverStatus** command:

-quiet
Suppresses the progress information that the **serverStatus** command prints in normal mode.

-logfile <fileName>
Specifies the location of the log file to which information gets written.

-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. The default for this option is the default profile.

-replacelog
Replaces the log file instead of appending to the current log.

-trace
Generates trace information into a file for debugging purposes.

-username <name>
Specifies the user name for authentication if security is enabled. Acts the same as the -user option.

-user <name>
Specifies the user name for authentication if security is enabled. Acts the same as the -username option.

-password <password>
Specifies the password for authentication if security is enabled.

-help
Prints a usage statement.

-? Prints a usage statement.

Usage scenario

The following examples demonstrate correct syntax:

```
serverStatus server1

serverStatus -all (returns status for all defined servers)

serverStatus -trace (produces the serverStatus.log file)
```

removeNode command

The **removeNode** command returns a node from a Network Deployment distributed administration cell to a base WebSphere Application Server installation. For more information about where to run this command, see the Using command tools article.

The **removeNode** command only removes the node-specific configuration from the cell. This command does not uninstall any applications that were installed as the result of executing an **addNode** command. Such applications can subsequently deploy on additional servers in the Network Deployment cell. As a consequence, an **addNode** command with the -includeapps option executed after a **removeNode** command does not move the applications into the cell because they already exist from the first **addNode** command. The resulting application servers added on the node do not contain any applications. To deal with this situation, add the node and use the deployment manager to manage the applications. Add the applications to the servers on the node after it is incorporated into the cell.

The **removeNode** command does the following:

- Stops all of the running server processes in the node, including the node agent process.
- Removes the configuration documents for the node from the cell repository by sending commands to the deployment manager.
- Copies the original application server cell configuration into the active configuration.

Depending on the size and location of the new node you remove from the cell, this command can take a few minutes to complete.

Syntax

The command syntax is as follows:

```
removeNode [options]
```

All arguments are optional.

Parameters

The following options are available for the **removeNode** command:

-quiet
Suppresses the progress information that the **removeNode** command prints in normal mode.

-logfile <fileName>
Specifies the location of the log file to which information is written.

-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. The default for this option is the default profile.

-replacelog
Replaces the log file instead of appending to the current log.

-trace
Generates trace information into a file for debugging purposes.

-statusport <portNumber>
Specifies that an administrator can set the port number for the node agent status callback.

-username <name>
Specifies the user name for authentication if security is enabled. Acts the same as the -user option.

-user <name>
Specifies the user name for authentication if security is enabled. Acts the same as the -username option.

-password <password>
Specifies the password for authentication if security is enabled.

-force
Cleans up the local node configuration regardless of whether you can reach the deployment manager for cell repository cleanup. After using the `-force` parameter, you may need to use the **cleanupNode** command on the deployment manager.

-help
Prints a usage statement.

-? Prints a usage statement.

Usage scenario

The following examples demonstrate correct syntax:

```
removeNode -quiet

removeNode -trace (produces the removeNode.log file)
```

cleanupNode command

The **cleanupNode** command cleans up a node configuration from the cell repository. Only use this command to clean up a node if you have a node defined in the cell configuration, but the node no longer exists. For more information about where to run this command, see the Using command tools article.

Syntax

The command syntax is as follows:

```
cleanupNode <node name> <deploymgr host> <deploymgr port> [options]
```

where the first argument is required.

Parameters

The following options are available for the **cleanupNode** command:

-quiet
Suppresses the progress information that the **cleanupNode** command prints in normal mode.

-trace
Generates trace information into a file for debugging purposes.

-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. The default for this option is the default profile.

Usage scenario

The following examples demonstrate correct syntax:

```
cleanupNode myNode
cleanupNode myNode -trace
```

syncNode command

The **syncNode** command forces a configuration synchronization to occur between the node and the deployment manager for the cell in which the node is configured.

The node agent server runs a configuration synchronization service that keeps the node configuration synchronized with the master cell configuration. If the node agent is unable to run because of a problem in the node configuration, you can use the **syncNode** command to perform a synchronization when the deployment manager is not running in order to force the node configuration back in sync with the cell configuration.

For more information about where to run this command, see the Using command tools article.

Syntax

The command syntax is as follows:

```
syncNode <deploymgr host> <deploymgr port> [options]
```

where the `<deploymgr host>` argument is required.

Parameters

The following options are available for the **syncNode** command:

-stopservers
Tells the **syncNode** command to stop all servers on the node, including the node agent, before performing configuration synchronization with the cell.

-restart
Tells the **syncNode** command to launch the node agent process after configuration synchronization completes.

-nowait
Tells the **syncNode** command not to wait for successful initialization of the launched node agent process.

-quiet
Suppresses the progress information that the **syncNode** command prints in normal mode.

-logfile <fileName>
Specifies the location of the log file to which information gets written.

-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. The default for this option is the default profile.

-replacelog
Replaces the log file instead of appending to the current log.

-trace
Generates trace information into a file for debugging purposes.

-timeout <seconds>
Specifies the waiting time before node agent initialization times out and returns an error.

-statusport <portnumber>
Specifies that an administrator can set the port number for node agent status callback.

-username <name>
Specifies the user name for authentication if security is enabled. Acts the same as the -user option.

-user <name>
Specifies the user name for authentication if security is enabled. Acts the same as the -username option.

-password <password>
Specifies the password for authentication if security is enabled.

-conntype <type>
Specifies the Java Management Extensions (JMX) connector type to use for connecting to the deployment manager. Valid types are Simple Object Access Protocol (SOAP) or Remote Method Invocation (RMI).

-help
Prints a usage statement.

-? Prints a usage statement.

Usage scenario

The following examples demonstrate correct syntax:

```
syncNode testhost 8879

syncNode deploymgr 8879 -trace (produces the syncNode.log file)

syncNode host25 4444 -stopservers -restart (assumes that the deployment manager JMX port is 4444)
```

backupConfig command

The **backupConfig** command is a simple utility to back up the configuration of your node to a file. By default, all servers on the node stop before the backup is made so that partially synchronized information is not saved. For more information about where to run this command, see the Using command tools article. If you do not have root authority, you must specify a path for the backup file in a location where you have write permission. The backup file will be in zip format and a `.zip` extension is recommended.

Syntax

The command syntax is as follows:

```
backupConfig <backup_file> [options]
```

where *backup_file* specifies the file to which the backup is written. If you do not specify one, a unique name is generated.

Parameters

The following options are available for the **backupConfig** command:

-nostop
Tells the **backupConfig** command not to stop the servers before backing up the configuration.

-quiet
Suppresses the progress information that the **backupConfig** command prints in normal mode.

-logfile <fileName>
Specifies the location of the log file to which information gets written.

-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. The default for this option is the default profile.

-replacelog
Replaces the log file instead of appending to the current log.

-trace
Generates trace information into the log file for debugging purposes.

-username <name>
Specifies the user name for authentication if security is enabled in the server. Acts the same as the -user option.

-user <name>
Specifies the user name for authentication if security is enabled in the server. Acts the same as the -username option.

-password <password>
Specifies the password for authentication if security is enabled in the server.

-help
Prints a usage statement.

-? Prints a usage statement.

Usage scenario

The following example demonstrates correct syntax:

```
backupConfig
```

This example creates a new file that includes the current date. For example: `WebSphereConfig_2003-04-22.zip`

```
backupConfig myBackup.zip -nostop
```

This example creates a file called `myBackup.zip`, and does not stop any servers before beginning the backup process.

restoreConfig command

The **restoreConfig** command is a simple utility to restore the configuration of your node after backing up the configuration using the **backupConfig** command. By default, all servers on the node stop before the configuration restores so that a node synchronization does not occur during the restoration. If the configuration directory already exists, it is renamed before the restoration occurs. For more information about where to run this command, see the Using command tools article.

For AIX only, if you are using a logical directory for `was_install/config`, the **restoreConfig** command will not work.

Syntax

The command syntax is as follows:

```
restoreConfig <backup_file> [options]
```

where *backup_file* specifies the file to be restored. If you do not specify one, the command will not run.

Parameters

The following options are available for the **restoreConfig** command:

-nowait
Tells the **restoreConfig** command not to stop the servers before restoring the configuration.

-quiet
Suppresses the progress information that the **restoreConfig** command prints in normal mode.

-location <directory_name>
Specifies the directory where the backup file is restored. The location defaults to the `install_root/config` directory.

-logfile <fileName>
Specifies the location of the log file to which information gets written.

-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. The default for this option is the default profile.

-replacelog
Replaces the log file instead of appending to the current log.

-trace
Generates trace information into the log file for debugging purposes.

-username <name>
Specifies the user name for authentication if security is enabled in the server. Acts the same as the -user option.

-user <name>
Specifies the user name for authentication if security is enabled in the server. Acts the same as the -username option.

-password <password>
Specifies the password for authentication if security is enabled in the server.

-help
Prints a usage statement.

-? Prints a usage statement.

You can use the **restoreConfig** command to recover an application server if it fails. Perform the following steps:

1. Locate the automatic migration backup in the `install_root/temp` directory. For example: `MigrationBackup.Thu-Aug-28-10.15045-2003.zip`
2. Restore the configuration with the **restoreConfig** command. For example: `restoreConfig install_root/temp/MigrationBackup.Thu-Aug-28-10.15045-2003.zip`

Usage scenario

The following example demonstrates correct syntax:

```
restoreConfig WebSphereConfig_2003-04-22.zip
```

The following example restores the given file to the `/tmp` directory and does not stop any servers before beginning the restoration:

```
restoreConfig WebSphereConfig_2003-04-22.zip -location /tmp -nostop
```

Be aware that if you restore the configuration to a directory that is different from the directory that was backed up when you performed the **backupConfig** command, you may need to manually update some of the paths in the configuration directory.

EARExpander command

Use the **EARExpander** command to expand an enterprise archive file (EAR) into a directory to run the application in that EAR file. You can collapse a directory containing application files into a single EAR file.

You can type `EARExpander` with no arguments to learn more about its options. For more information about where to run this command, see the Using command tools article.

Syntax

The command syntax is as follows:

```
EarExpander -ear earName -operationDir dirName -operation <expand | collapse> [-expansionFlags <all|war>]
```

Parameters

The following options are available for the **EARExpander** command:

-ear
Specifies the name of the input EAR file for the expand operation or the name of the output EAR file for the collapse operation.

-operationDir
Specifies the directory where the EAR file is expanded or specifies the directory from where files are collapsed.

-operation <expand | collapse>
The `expand` value expands an EAR file into a directory structure required by the WebSphere Application Server run time. The `collapse` value creates an EAR file from an expanded directory structure.

-expansionFlags <all | war>
(Optional) The `all` value expands all files from all of the modules. The `war` value only expands the files from Web archive file (WAR) modules.

-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. The default for this option is the default profile.

Usage scenario

The following examples demonstrate correct syntax:

```
EARExpander -ear C:\WebSphere\AppServer\installableApps\DefaultApplication.ear
-operationDir  C:\MyApps -operation expand -expansionFlags war

EARExpander -ear C:\backup\DefaultApplication.ear
-operationDir C:\MyAppsDefaultApplication.ear -operation collapse
```

GenPluginCfg command

This topic describes the command-line syntax for the GenPluginCfg command. This command is used to regenerate the WebSphere Web server plug-in configuration file, plugin-cfg.xml. For more information about where to run this command, see the Using command tools article.

CAUTION:
Regenerating the plug-in configuration can overwrite manual configuration changes that you might want to preserve. Before performing this task, understand its implications as described in "Communicating with Web servers" on page 57.

To regenerate the plug-in configuration, you can either click on **Servers > Web Servers** in the administrative console, select a Web server and then click Generate Plug-in, or you can issue the following command:

```
GenPluginCfg.sh|bat
```

Both methods for regenerating the plug-in configuration create a `plugin-cfg.xml` file in ASCII format, which is the proper format for execution in a distributed environment.

You can use the -profileName option to define 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. The default for this option is the default profile.

Syntax

The command syntax is as follows:

```
GenPluginCfg [[-option.name optionValue]...]
```

When the GenPluginCfg command is issued with the option -webserver.name webservrName, wsadmin generates a plug-in configuration file for the Web server. This settings in this generated configuration file are based on the list of applications that are deployed on the Web server. When this command is issued without the option -webserver.name webservrName, the plug-in configuration file is generated based on topology.

Parameters

The following options are available for the **startServer** command:

-config.root configroot_dir
: Defaults to environment variable CONFIG_ROOT.

-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. The default for this option is the default profile.

-cell.name cell
: Defaults to environment variable WAS_CELL.

-node.name node
: Defaults to environment variable WAS_NODE.

-webserver.name webserver1
: Required for creating plug-in configuration file for a given Web server.

-propagate yes/no
: Applicable only when the option webserver.name is specified. Defaults to no.

-cluster.name cluster1,cluster2 | ALL
: Optional list of clusters. Ignored when the option webserver.name is specified.

-server.name server1,server2
: Optional list of servers. Required for single server plug-in generation. Ignored when the option webserver.name is specified.

-output.file.name file_name
: Defaults to the configroot_dir/plugin-cfg.xml file. Ignored when the option webserver.name is specified.

-destination.root root
: Installation root of the machine configuration is used on. Ignored when the option webserver.name is specified.

-destination.operating.system windows/unix
: Operating system of the machine configuration is used on. Ignored when the option webserver.name is specified.

-debug yes/no
: Defaults to no.

-help
Prints a usage statement.

-? Prints a usage statement.

Usage scenario

To generate a plug-in configuration for all of the clusters in a cell:

```
GenPluginCfg -cell.name NetworkDeploymentCell
```

To generate a plug-in configuration for a single server:

```
GenPluginCfg -cell.name BaseApplicationServerCell -node.name appServerNode -server.name appServerName
```

To generate a plug-in configuration file for a Web server:

```
GenPluginCfg -cell.name BaseApplicationServerCell -node.name webserverNode -webserver.name webserverName
```

Chapter 5. 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.

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” on page 669 for details and variations

```
startServer server
```

where *server* is the application server that you want to start.

Stopping the servers

Use the same command as to start, except substitute stop for start. For example, to stop an application server, issue the command:

```
stopServer server
```

- Use administrative clients and tools.

Quick reference: Opening the administrative console

To open the console, 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 stopping the administrative console” on page 171 for details, as needed.

  - To launch a scripting client, see “Starting the wsadmin scripting client” on page 250.
  - To learn about all available administrative clients, see Chapter 4, “Using the administrative clients,” on page 171.
  - For performance monitoring, see ″Monitoring performance with Tivoli Performance Viewer (TPV)″ in the information center.

  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 874.

Text editor

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

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

  Launchpad

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

  See ″Using the launchpad to start the installation″ in the information center.

  First Steps

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

  See “firststeps command” on page 33.

  Profile creation tool

  The installation places the product binary files on a machine and creates a default profile.

- Use troubleshooting tools - see ″Working with troubleshooting tools″ in the information center.

Chapter 6. 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 702 refers to additional sources.

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

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

Class loaders

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.

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 contains the Java 2 Platform, Enterprise Edition (J2EE) application programming interfaces (APIs) of the WebSphere Application Server product in the `j2ee.jar` file. Because the J2EE APIs are in this class loader, you can add libraries that depend on the J2EE APIs to the class path system property to extend a server class path. However, a preferred method of extending a server class path is to add a shared library.

2. A WebSphere-specific extensions class loader

   The WebSphere Application Server extensions class loader loads the WebSphere Application Server run-time and J2EE 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. The WebSphere Application Server run time enables you to associate a shared library class path with an application.

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 consist of EJB modules, dependency JAR files, resource adapters, and 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.

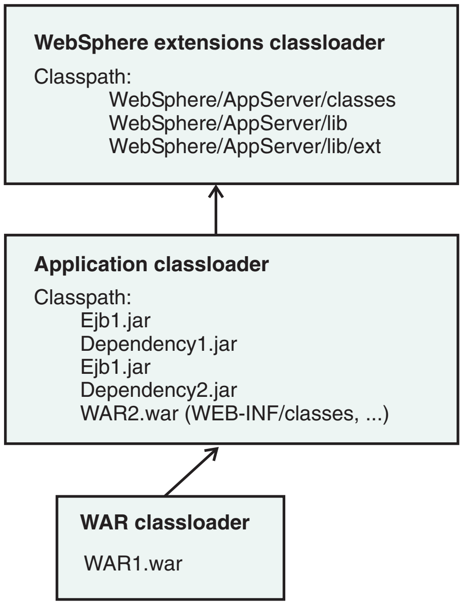
Note: WebSphere Application Server class loaders never load application client modules.

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.

This application class loader can load the Web modules for each application if the class-loader policy of that WAR module is also set to `Application`. If the class-loader policy of the WAR module is set to `Application`, then the application loader loads the WAR module classes. If the WAR class-loader policy is set to `Module`, then each WAR module receives its own class loader.

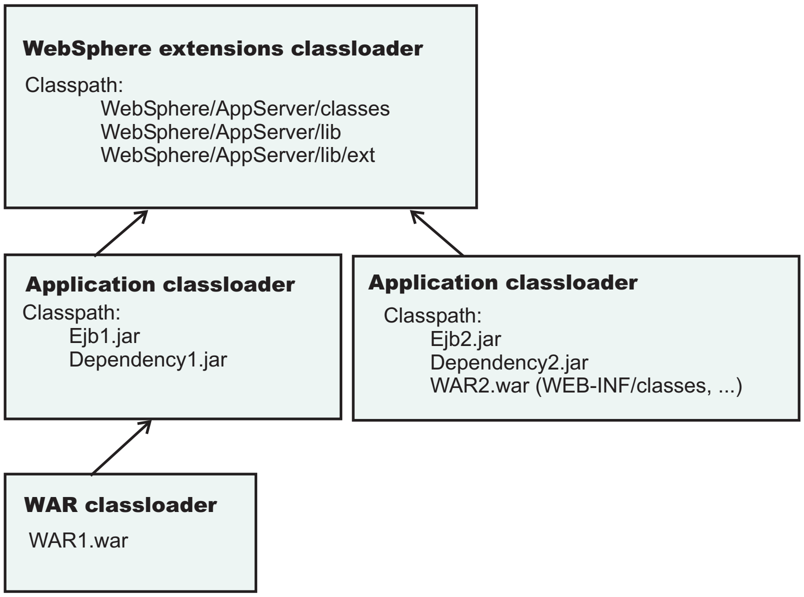
The following example shows that when the application class-loader policy is set to `Single`, a single application class loader loads all of the EJB modules, dependency JAR files, and shared libraries of all applications on the server. The single application class loader can also load Web modules if 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.

```
Application class-loader policy: Single

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.

```
Application class-loader policy: Multiple

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

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

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

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

3. Specify the application class-loader mode for the application server. The application class loading mode specifies the class-loader mode when the application class-loader policy is `Single`. On the settings page for the application server, select either of the following values:

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

4. Specify the class-loader mode for the class loader.
   a. On the settings page for the application server, click **Java and Process Management > Class loader** to access the Class loader page.
   b. On the Class loader page, click **New** to access the settings page for a class loader.
   c. On the settings page for a class loader, specify the class-loader 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 723 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.
  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.
- 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**.
  – 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**.

  After you change the JVM system property, click **OK**, save your administrative configuration, stop the application server, and then restart the application server.
- 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 the information center 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^{TM} Platform Specification
- Sun's J2EE^{TM} Extension Mechanism Architecture

Chapter 7. 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

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.

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
   - Java administrative programs that use JMX APIs
   - Java programs that define a J2EE DeploymentManager object in accordance with J2EE Deployment API Specification (JSR-88)
3. Start the deployed application files using
   - Administrative console
   - wsadmin startApplication
   - Java programs that use ApplicationManager or AppManagement MBeans
   - Java programs that define a J2EE DeploymentManager object in accordance with J2EE Deployment API Specification (JSR-88)

Save the changes to your administrative configuration.

Next, test the application. For example, point a Web browser at the URL for a deployed application 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 on a WebSphere Application Server Version 5 product.

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

Table 6. 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.

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

Table 7. Ways to install application files

Option	Method	Modules	Comments	Starting after install

Table 7. Ways to install application files (continued)

Administrative console install wizard See “Installing application files with the console” on page 709.	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 714 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.	All EAR, EJB, WAR, RAR, and application client files	″Getting started with scripting″ in the information center provides an overview of wsadmin.	• Invoke the AdminApp *startApplication* command. • Invoke the *startApplication* method on an ApplicationManager MBean using AdminControl. .
Java application programming interfaces	Install programs by completing the steps in ″Managing applications through programming″ in the information center.	All EAR files	Use MBeans to install the application.	Start the application by calling the *startApplication* method on a proxy.
WebSphere rapid deployment Refer to articles under **Rapid deployment of J2EE applications** in the 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.

Table 7. Ways to install application files (continued)

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 714 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 719 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`.

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.

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.

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.
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.

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

After the application file installs successfully, do the following:

1. Associate any shared libraries that the application needs to the application.
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 a single-server installation, application files are copied to the destination directory when the changes are saved.
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. 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 server 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. After application installation, you can modify the bindings as needed using the administrative console.

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

1. Go to the Preparing for application install pages.

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

   For this example, the base file name is `my_appl.ear` and the file resides on a server at `C:\sample_apps`.
3. 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 stand-alone 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.

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

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. An application with an unavailable status might, in fact, be running but have an unavailable status because the server running the administrative console cannot communicate with the server running the application.
?	**Not applicable**	Application does not provide information as to whether it is running.

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. Application names must be unique within a cell and cannot contain characters that are not allowed in object names.

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 `${MY_APPS}`. You can use a pathmap variable in any installation. A WebSphere Application Server variable `${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.

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 or Web servers onto which to install modules of your application.
- 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.
     – Session management

- Libraries or library references
- Target mappings
- Deployment descriptors
- Publish WSDL files
- Provide JMS and EJB endpoint URL information
- Provide HTTP endpoint URL information
- Map security roles to users/groups
- Provide JNDI Names for Beans. For more information, refer to “Task overview: Using enterprise beans in applications” on page 817.
- Map resource references to resources
- Map EJB references to beans. For more information, refer to “Task overview: Using enterprise beans in applications” on page 817.
- 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 1171.
- Map virtual hosts for web modules. For more information, refer to ″Configuring virtual hosts″ in the information center.
- Map modules to servers
- 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.

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. 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 references panel.

Virtual host bindings for Web modules

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

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

During and after application installation, you can map a virtual host to a Web module defined in your application. On the Map virtual hosts for Web modules panel, specify a virtual host. The port number specified in the virtual host definition is used in the URL that is used to access artifacts such as servlets and JSP files in the Web module. For example, an external URL for a Web artifact such as a JSP file is `http://`*`host_name`*`:`*`virtual_host_port`*`/`*`context_root`*`/`*`jsp_path`*. After installation, click **Applications > Enterprise Applications >***application_name* **> 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. 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″ in the information center.

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 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 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.

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
- 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.

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.
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.

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

Target

States the name of the target server 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 on a WebSphere Application Server Version 5 product.

A *6.x deployment target* is a server 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 is started, stopped or unavailable.

Target mapping settings

Use this page to map a deployed application or module to a server.

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

Target:

States the name of the target server 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 739 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. You can deploy J2EE 1.4 modules to Version 6.x servers only. You cannot deploy J2EE 1.4 modules to servers on Version 5.x nodes. Refer to “Installable module versions” on page 706 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 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. 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 server 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. 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 only.

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

1. Save the changes to your configuration.

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

Ways to update application files

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

Table 8. Ways to update application files

Option	Method	Comments	Starting after update
Administrative console update wizard See "Updating applications" on page 736.	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″ in the information center provides an overview of wsadmin.	Invoke the wsadmin *startApplication* command.

Table 8. Ways to update application files (continued)

Java application programming interfaces. See ″Using administrative programs (JMX)″ in the information center.	Update deployed applications by completing the steps in ″Managing applications through programming″ in the information center.	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. 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, 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:

- For files added or updated at 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.
- For files 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.
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 736, changes made using hot deployment or dynamic reloading do not use the administrative console or a wsadmin scripting command. You must directly manipulate the application files on the server where the application is deployed.

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

1. Locate your expanded application files. The application files are in the directory you specified when installing the application or, if you did not specify a custom target directory, are in the default target directory, *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. **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.
4. Change or add the following components or modules as needed:
   - Application files
   - WAR files
   - EJB Jar files
   - HTTP plug-in configuration files
5. For changes to take effect, you might need to start, stop, or restart an application. "Starting and stopping applications" on page 733 provides information on using the administrative console to start, stop, or restart an application. ″Starting applications with scripting″ and ″Stopping applications with scripting″ provide information on using the wsadmin scripting tool.

Changing or adding application files

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

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

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 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 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 for information on how to set this property.) You can also run the `GenPluginCfg.bat/sh` script, or issue a wsadmin command to regenerate the plug-in configuration file.

Hot deployment	Yes
Dynamic reloading	Yes

Uninstalling applications

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

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

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

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

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

Removing a file

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

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

- Remove a file from an application.
  1. Go to the Enterprise Applications page. Click **Applications > Enterprise Applications** in the console navigation tree.
  2. Select the application that contains a file you want removed.
  3. Click **Remove File**. The Remove a file 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. In the single-server product, application binaries are deleted after you save the changes.

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 the information center 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^{TM} Tutorial: The Duke's Bookstore Application
- Best Practices in WebSphere Application: Separating the developers from the administrators
- Designing Enterprise Applications with the Java^{TM} 2 Platform, Enterprise Edition, Second Edition
- Designing Enterprise Applications, Second Edition
- Building Java^{TM} 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^{TM} 2 Platform, Enterprise Edition

Administration
- Listing of all IBM WebSphere Application Server Redbooks

Chapter 8. Developing 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. Also use this section to learn about developing WebSphere applications.

See Learn about WebSphere applications: Overview and new features 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	See the *Securing WebSphere applications* PDF	See the *Administering applications* PDF	See the *Securing WebSphere applications* PDF	See the *Securing WebSphere applications* PDF
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 the ″Resources for learning″ article for links to design documentation.

3. Develop the Web application, using WebSphere Application Server extensions to enhance its functionality.
4. Assemble the Web application into a Web module using an assembly tool. Web module assembly properties might include the ability to:
   - Configure servlet page lists.
   - Configure servlet filters.
   - Serve servlets by class name.
   - Enable file serving.
5. Deploy the Web module or application module that contains the Web application.

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

Web applications

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

The files in a Web application are related in that they work together to perform a business logic function. For example, one of the WebSphere Application Server samples is a *Simple Greeting* Web application. This application, comprised of a servlet and Web pages, greets new users when 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
- Perferred 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. If your JSP files do use WebSphere public APIs within scriptlets, then add those additional JAR files to the jspCompileClasspath option. The entries are separated by spaces, and are assumed to be relative to the WebSphere Application Server installation root.

  1. A space-separated list of JAR files, relative to the WebSphere installation directory. This instructs the JSP engine to use the small class path, with the additional named JAR files.

  2. No value at all. This instructs the JSP engine to use the small class path, without any additional user-defined JAR files. See the example in ″Configuring JSP engine parameters″ in the information center.

  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 files). 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**

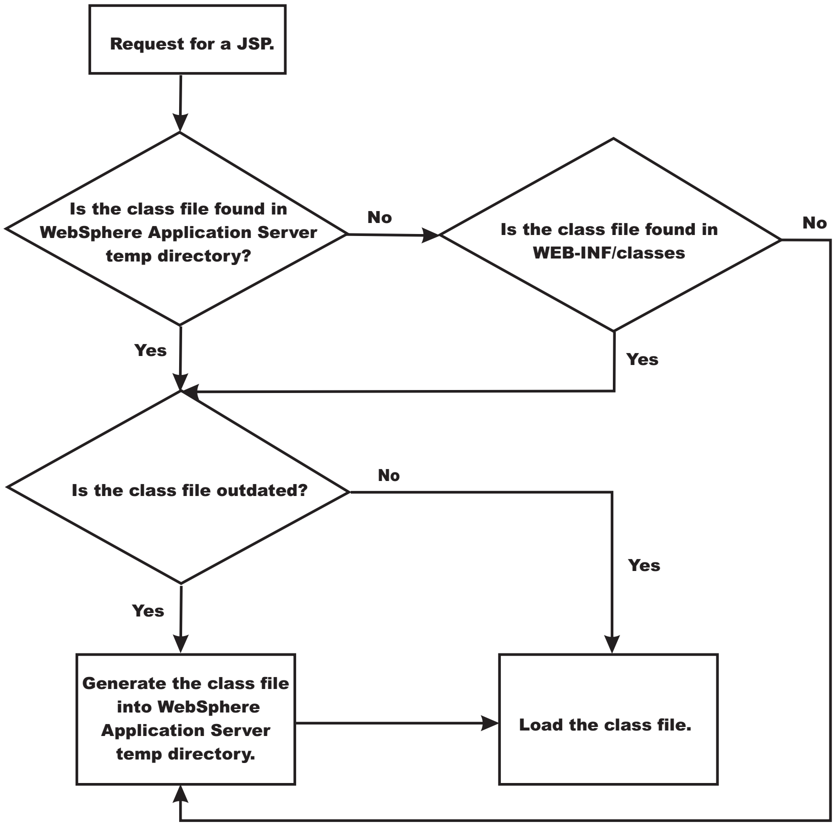
  *When thread-level tag handler pooling is used, tag handlers may be reused among separate occurrences of a custom action across all JSP pages in a single Web module across separate requests. The default for this parameter is `false`. This parameter requires regeneration of Java source.
- **verbose**

  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_ROOT}/profiles/profilename/temp`. 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.


Request for a JSP.
Is the class file found in WebSphere Application Server temp directory?
No
Is the class file found in WEB-INF/classes
No
Yes
Yes
Is the class file outdated?
No
Yes
Yes
Generate the class file into WebSphere Application Server temp directory.
Load the class file.

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 766 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**	**useFullPackage Names=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 runtime reloading 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?	useFullPackage Names

<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.

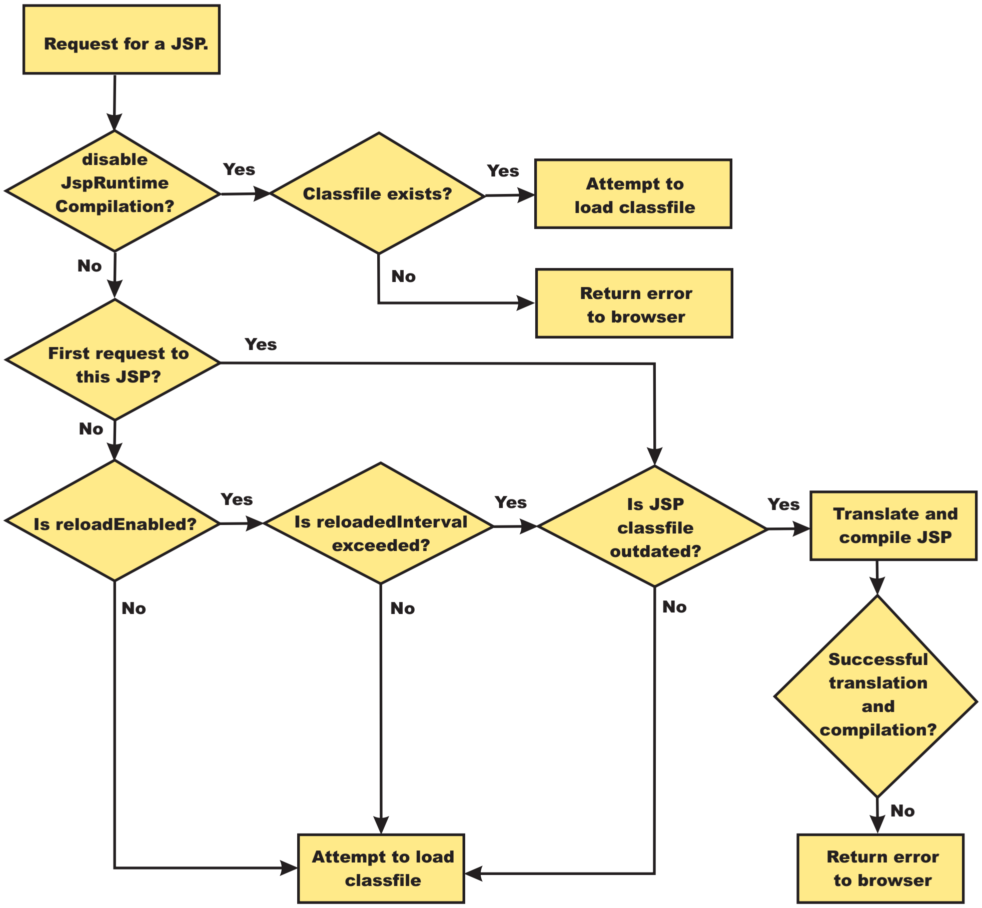

Request for a JSP.
disable JspRuntime Compilation?
Yes
Classfile exists?
Yes
Attempt to load classfile
No
No
Return error to browser
First request to this JSP?
Yes
No
Is reloadEnabled?
Yes
Is reloadedInterval exceeded?
Yes
Is JSP classfile outdated?
Yes
Translate and compile JSP
No
No
No
Successful translation and compilation?
No
Attempt to load classfile
Return error to browser

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.

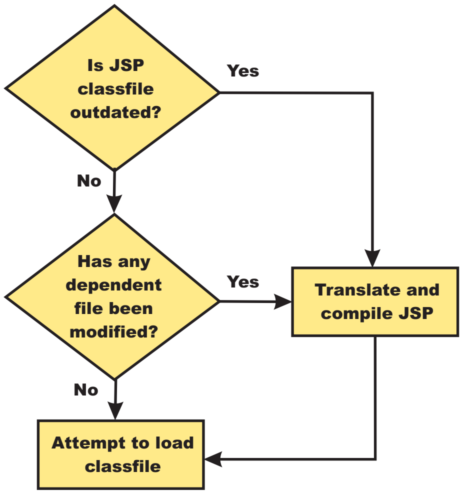

Is JSP classfile outdated?
Yes
No
Has any dependent file been modified?
Yes
Translate and compile JSP
No
Attempt to load classfile

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 as described in the "Assembling applications" topic in the information center.

  Valid values for this setting are `true` or `false`. If this parameter is set to `true`, then, for that specific Web application, translation and compilation of the JSP files is disabled at run time, and the JSP engine only loads precompiled files.

- If neither the Web container custom property nor the JSP parameter is set, the first request for a JSP file results in the translation and compilation of the JSP file when the `.class` file does not exist or is outdated. Subsequent requests for the file also result in translations and compilations, but only if the following conditions are met:
  - Translations are required because the `.class` file is outdated.
  - Reloading is enabled for the Web module.
  - Reload interval is exceeded.

If you disable run-time compilation and a request arrives for a JSP file that does not have a matching `.class` file, the JSP engine returns HTTP error 500 (Internal server error) to the browser. In this case, an exception is written to the System Out (SYSOUT) and First Failure Data Capture (FFDC) logs.

If a JSP file has a matching `.class` file but that file is out of date, the JSP engine still loads the `.class` file into memory.

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 760.

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 771, 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 Hype rText 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.

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.

View links to additional information about:
- “Web applications: Resources for learning” on page 791
- “Web applications: Resources for learning” on page 791
- “Web applications: Resources for learning” on page 791

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^{TM} 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 and session recovery.
2. Create or modify your own applications to use session support to maintain sessions on behalf of Web applications.
3. Assemble your application. See ″Assembling applications″ in the information center.
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 failover facility. When an application server receives a request associated with a session ID that it currently does not have in memory, it can obtain the required session state by accessing the external store (database or memory-to-memory). If distributed session support is not enabled, an application server cannot access session information for HTTP requests that are sent to servers other than the one where the session was originally created. Session Management implements caching optimizations to minimize the overhead of accessing the external store, especially when consecutive requests are routed to the same application server.

Storing session states in an external store also provides a degree of fault tolerance. If an application server goes offline, the state of its current sessions is still available in the external store. This availability enables other application servers to continue processing subsequent client requests associated with that session.

Saving session states to an external location does not completely guarantee their preservation in case of a server failure. For example, if a server fails while it is modifying the state of a session, some information is lost and subsequent processing using that session can be affected. However, this situation represents a very small period of time when there is a risk of losing session information.

The drawback to saving session states in an external store is that accessing the session state in an external location can use valuable system resources. session management can improve system performance by caching the session data at the server level. Multiple consecutive requests that are directed to the same server can find the required state data in the cache, reducing the number of times that the actual session state is accessed in external store and consequently reducing the overhead associated with external location access.

Session tracking options

There are several options for session tracking, depending on what sort of tracking method you want to use:

- Session tracking with cookies
- Session tracking with URL rewriting
- Session tracking with Secure Sockets Layer (SSL) information

Session tracking with cookies: Tracking sessions with cookies is the default. No special programming is required to track sessions with cookies.

Session tracking with URL rewriting: An application that uses URL rewriting to track sessions must adhere to certain programming guidelines. The application developer needs to do the following:
- Program servlets to encode URLs
- Supply a servlet or JavaServer Pages (JSP) file as an entry point to the application

Using URL rewriting also requires that you enable URL rewriting in the session management facility.

Note: In certain cases, clients cannot accept cookies. Therefore, you cannot use cookies as a session tracking mechanism. Applications can use URL rewriting as a substitute.

Program session servlets to encode URLs

Depending on whether the servlet is returning URLs to the browser or redirecting them, include either the encodeURL method or the encodeRedirectURL method in the servlet code. Examples demonstrating what to replace in your current servlet code follow.

Rewrite URLs to return to the browser

Suppose you currently have this statement:

```
out.println("<a href=\"/store/catalog\">catalog<a>");
```

Change the servlet to call the encodeURL method before sending the URL to the output stream:

```
out.println("<a href=\"");
out.println(response.encodeURL ("/store/catalog"));
out.println("\">catalog</a>");
```

Rewrite URLs to redirect

Suppose you currently have the following statement:

```
response.sendRedirect ("http://myhost/store/catalog");
```

Change the servlet to call the encodeRedirectURL method before sending the URL to the output stream:

```
response.sendRedirect (response.encodeRedirectURL ("http://myhost/store/catalog"));
```

The encodeURL method and encodeRedirectURL method are part of the HttpServletResponse object. These calls check to see if URL rewriting is configured before encoding the URL. If it is not configured, the calls return the original URL.

If both cookies and URL rewriting are enabled and the response.encodeURL method or encodeRedirectURL method is called, the URL is encoded, even if the browser making the HTTP request processed the session cookie.

You can also configure session support to enable protocol switch rewriting. When this option is enabled, the product encodes the URL with the session ID for switching between HTTP and HTTPS protocols.

Supply a servlet or JSP file as an entry point

The entry point to an application, such as the initial screen presented, may not require the use of sessions. However, if the application in general requires session support (meaning some part of it, such as a servlet, requires session support), then after a session is created, all URLs are encoded to perpetuate the session ID for the servlet (or other application component) requiring the session support.

The following example shows how you can embed Java code within a JSP file:

```
<%
response.encodeURL ("/store/catalog");
%>
```

Session tracking with SSL information: No special programming is required to track sessions with Secure Sockets Layer (SSL) information.

To use SSL information, turn on **Enable SSL ID tracking** in the session management property page. Because the SSL session ID is negotiated between the Web browser and HTTP server, this ID cannot survive an HTTP server failure. However, the failure of an application server does not affect the SSL session ID if an external HTTP server is present between WebSphere Application Server and the browser.

SSL tracking is supported for the IBM HTTP Server and iPlanet Web servers only. You can control the lifetime of an SSL session ID by configuring options in the Web server. For example, in the IBM HTTP Server, set the configuration variable SSLV3TIMEOUT to provide an adequate lifetime for the SSL session ID. An interval that is too short can cause a premature termination of a session. Also, some Web browsers might have their own timers that affect the lifetime of the SSL session ID. These Web browsers may not leave the SSL session ID active long enough to serve as a useful mechanism for session tracking. The internal HTTP Server of WebSphere Application Server also supports SSL tracking.

When using the SSL session ID as the session tracking mechanism in a cloned environment, use either cookies or URL rewriting to maintain session affinity. The cookie or rewritten URL contains session affinity information that enables the Web server to properly route a session back to the same server for each request.

Session recovery support

For session recovery support, WebSphere Application Server provides distributed session support in the form of database sessions. Use session recovery support under the following conditions:

- When the user’s session data must be maintained across a server restart
- When the user’s session data is too valuable to lose through an unexpected server failure

All the attributes set in a session must implement java.io.Serializable if the session requires external storage. In general, consider making all objects held by a session serialized, even if immediate plans do not call for session recovery support. If the Web site grows, and session recovery support becomes necessary, the transition occurs transparently to the application if the sessions only hold serialized objects. If not, a switch to session recovery support requires coding changes to make the session contents serialized.

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.

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

Feature or option	Goal	Applies to sessions in memory, database, or memory-to-memory
Tablespace and row size	Increase efficiency of write operations to database.	Database (DB2 only)

Base in-memory session pool size: The base in-memory session pool size number has different meanings, depending on session support configuration:

- With in-memory sessions, session access is optimized for up to this number of sessions.

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

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

Overflow in non-distributed sessions

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

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

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

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

Tuning parameter settings:

Use this page to set tuning parameters for distributed sessions.

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

Tuning level:

Specifies that the session management facility provides certain predefined settings that affect performance.

Select one of these predefined settings or customize a setting. To customize a setting, select one of the predefined settings that comes closest to the setting desired, click **Custom settings**, make your changes, and then click **OK**.

Very high (optimize for performance)

Write frequency	Time based
Write interval	300 seconds

Write contents	Only updated attributes
Schedule sessions cleanup	true
First time of day default	0
Second time of day default	2

High

Write frequency	Time based
Write interval	300 seconds
Write contents	All session attributes
Schedule sessions cleanup	false

Medium

Write frequency	End of servlet service
Write contents	Only updated attributes
Schedule sessions cleanup	false

Low (optimize for failover)

Write frequency	End of servlet service
Write contents	All session attributes
Schedule sessions cleanup	false

Custom settings

Write frequency default	Time based
Write interval default	10 seconds
Write contents default	All session attributes
Schedule sessions cleanup default	false

Tuning parameter custom settings:

Use this page to customize tuning parameters for distributed sessions.

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

Write frequency:

Specifies when the session is written to the persistent store.

End of servlet service
A session writes to a database or another WebSphere Application Server instance after the servlet completes execution.

Manual update
A programmatic sync on the IBMSession object is required to write the session data to the database or another WebSphere Application Server instance.

Time based
Session data writes to the database or another WebSphere Application Server instance based on the specified Write interval value. Default: 10 seconds

Write contents:

Specifies whether updated attributes are only written to the external location or all of the session attributes are written to the external location, regardless of whether or not they changed. The external location can be either a database or another application server instance.

Only updated attributes	Only updated attributes are written to the persistent store.
All session attribute	All attributes are written to the persistent store.

Schedule sessions cleanup:

Specifies when to clean the invalid sessions from a database or another application server instance.

Specify distributed sessions cleanup schedule	Enables the scheduled invalidation process for cleaning up the invalidated HTTP sessions from the external location. Enable this option to reduce the number of updates to a database or another application server instance required to keep the HTTP sessions alive. When this option is not enabled, the invalidator process runs every few minutes to remove invalidated HTTP sessions. When this option is enabled, specify the two hours of a day for the process to clean up the invalidated sessions in the external location. Specify the times when there is the least activity in the application servers. An external location can be either a database or another application server instance.
First Time of Day (0 - 23)	Indicates the first hour during which the invalidated sessions are cleared from the external location. Specify this value as a positive integer between 0 and 23. This value is valid only when schedule invalidation is enabled.
Second Time of Day (0 - 23)	Indicates the second hour during which the invalidated sessions are cleared from the external location. Specify this value as a positive integer between 0 and 23. This value is valid only when schedule invalidation is enabled.

Best practices for using HTTP Sessions

- **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 handles special client request loads, in the administrative console, click **Servers** > **Application servers** >*server_name*. Then under Additional Properties, click **Thread Pools** to modify your thread pool settings.
3. If your application requires global settings for internal servlets for WAR files packaged by third-party tools, iIn the administrative console, click **Servers** > **Application servers** >*server_name* > **Web container settings** > **Web container**. Then under Additional Properties, click **Custom Properties** and enter the appropriate custom property.

Web container

A Web container handles requests for servlets, JavaServer Pages (JSP) files, and other types of files that include server-side code. The Web container creates servlet instances, loads and unloads servlets, creates and manages request and response objects, and performs other servlet management tasks.

The Web server plug-ins, provided by the WebSphere Application Server, help supported Web servers pass servlet requests to Web containers.

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 ModuleRef 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 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" on page 168, 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: 8 or 9.

HttpSessionIdLength
Use this property to configure the session identifier length. Do not use an extremely low value; using a low value results in reduced number of combinations possible, thereby increasing risk of guessing the session identifier. In a cluster, all cluster members should be configured with same ID length. Allowed range: 8 to 128. Default length: 23.

HttpSessionReaperPollInterval
Use this property to set a wake-up interval for the process that removes invalid sessions. Default is based on maximum inactive interval set in session management. Allowed value: integer.

NoAdditionalSessionInfo
Set this value to ″true″ to force removal of information that is not needed in session identifiers. In WebSphere Application Server base edition,a clone ID of -1 is never used; therefore, a clone ID is not included in base edition when this is set. Also, cache ID is not used with nonpersistent sessions; so the cache ID is not included with nonpersistent sessions when this value is set.

SessionIdentifierMaxLength
Use this value to set maximum length that a session identifier can grow. In a cluster, because of fail-over when a request goes to new cluster member, session management appends a new clone ID to the existing clone ID. In a large cluster, if for some reason servers are failing more often, then it is possible that the session identifier length can be more than expected reducing room for URL. So this property helps to find out the condition and take appropriate action to address servers fail-over. When this is specified, message is logged when specified maximum length is reached. Allowed value: integer.

SessionRewriteIdentifier
Use this property to change the key used with URL rewriting. Default key: jsessionid.

Configuring session tracking for Wireless Application Protocol (WAP) devices

Most Wireless Application Protocol (WAP) devices do not support cookies. The preferred way to track sessions for WAP devices is to use URL rewriting. However on most WAP devices, the maximum allowed URL length is 128 characters. With URL rewriting, a session indentifier is added to the URL itself, effectively decreasing the space available for the actual URL and the number of parameters that can be sent on a request.

To reduce the length of session identifier, you can configure key (jsessionid), session ID length and clone ID. To make these configuration changes, complete the following:

1. Open the Administrative console.
2. Click **Servers** > **Application Servers**.
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. 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 an existing database, 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.
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.
4. Go to the appropriate level of Session Management.
5. Click **Distributed Environment Settings**
6. Select and click **Database**.
7. Specify the Data Source JNDI name from step 2.
8. Specify the database user ID and password for accessing the database.
9. Retype the password for confirmation.
10. Configure a table space and page sizes for DB2 session databases.
11. Switch to a multirow schema.
12. Click **OK**.
13. If you want to change the tuning parameters, click **Custom Tuning Parameters** and select a setting or customize one.
14. Click **Apply**.
15. Click **Save**.

Switching to a multirow schema

By default, a single session maps to a single row in the database table used to hold sessions. With this setup, there are hard limits to the amount of user-defined, application-specific data that WebSphere Application Server can access.

1. Modify the Session Management facility properties to switch from single to multirow schema.
2. Manually drop the database table or delete all the rows in the database table that the product uses to maintain HttpSession objects.

   To drop the table:

   a. Determine which data source configuration Session Management is using.

b. In the data source configuration, look up the database name.
c. Use the database facilities to connect to the database.
d. Drop the SESSIONS table.

Configuring tablespace and page sizes for DB2 session databases

If you are using DB2 for session persistence, you can increase the page size to optimize performance for writing large amounts of data to the database. Page sizes of 8K, 16K, and 32K are supported.

To use a page size other than the default (4K), do the following:

1. If the SESSIONS table already exists, drop it from the DB2 database.
2. Create a new DB2 buffer pool and table space, specifying the same page size (8K, 16K or 32K) for both, and assign the new buffer pool to this table space.

   ```
   DB2 Connect to session
   DB2 CREATE BUFFERPOOL sessionBP SIZE 1000 PAGESIZE 8K
   DB2 Connect reset
   DB2 Connect to session
   DB2 CREATE TABLESPACE sessionTS PAGESIZE 8K MANAGED BY SYSTEM
       USING ('D:\DB2\NODE0000\SQL00005\sessionTS.0') BUFFERPOOL sessionBP
   DB2 Connect reset
   ```

   Refer to DB2 product documentation for details.
3. Configure the correct table space name and page size in the Session Management facility. Page size is referred to as *row size* on the Session Management page.)

When the product is restarted, the Session Management facility creates the new SESSIONS table in the specified tablespace based on the indicated page size.

Database settings

Use this page to specify the settings for database session support.

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

Datasource JNDI name:

Specifies the datasource description.

The JNDI name of the non-XA enabled 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.

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 1154 and “Messaging: Resources for learning” on page 1251.
2. Develop any enterprise beans that your application will use. See ″Developing enterprise beans″ in the information center.
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. See ″Assembling EJB modules″ in the information center for more information. This process includes setting security, see ″Securing enterprise bean applications″. For your EJB 2.x-compliant entity beans, you might also want to designate container-managed persistence (CMP) sequence groups. See ″Setting the run time for CMP sequence groups″ in the information center.
5. Assemble the modules into a J2EE application using the assembly tool. See ″Assembling applications″ in the information center.
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.
   - Debug access problems.
9. Assemble the production application using the assembly tools. See ″Assembling applications″ in the information center
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. See ″Assembling applications″ in the information center.
13. Tune the performance of the application. See ″Best practices for developing enterprise beans″ in the information center.

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.

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″ in the information center.

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

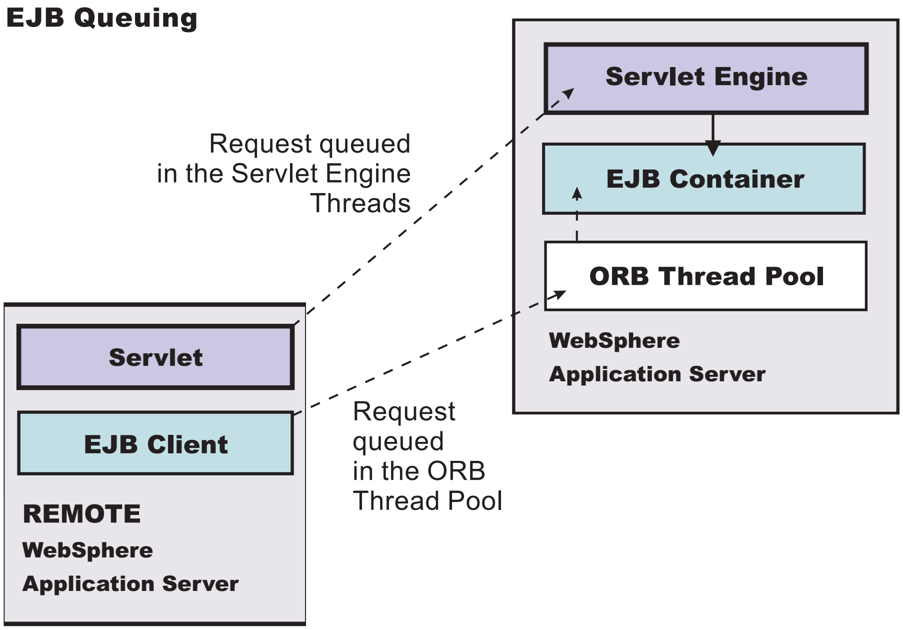
  You can download the specification from this URL.
- Java^{TM} 2 Platform: Compatibility with Previous Releases

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

EJB method Invocation Queuing

Method invocations to enterprise beans are only queued for remote clients, making the method call. An example of a remote client is an enterprise Java bean (EJB) client running in a separate Java virtual machine (JVM) (another address space) from the enterprise bean. In contrast, no queuing occurs if the EJB client, either a servlet or another enterprise bean, is installed in the same JVM on which the EJB method runs and on the same thread of execution as the EJB client.

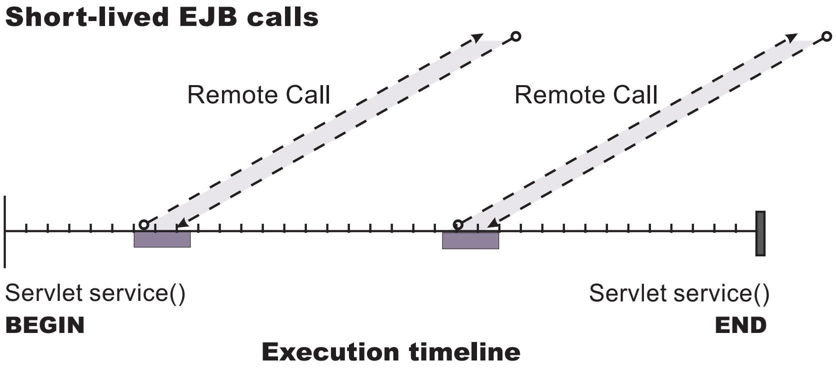
Remote enterprise beans communicate by using the Remote Method Invocation over Internet Inter-ORB Protocol (RMI-IIOP). Method invocations initiated over RMI-IIOP are processed by a server-side object request broker (ORB). The thread pool acts as a queue for incoming requests. However, if a remote method request is issued and there are no more available threads in the thread pool, a new thread is created. After the method request completes the thread is destroyed. Therefore, when the ORB is used to process remote method requests, the EJB container is an open queue, due to the use of unbounded threads. The following illustration depicts the two queuing options of enterprise beans.

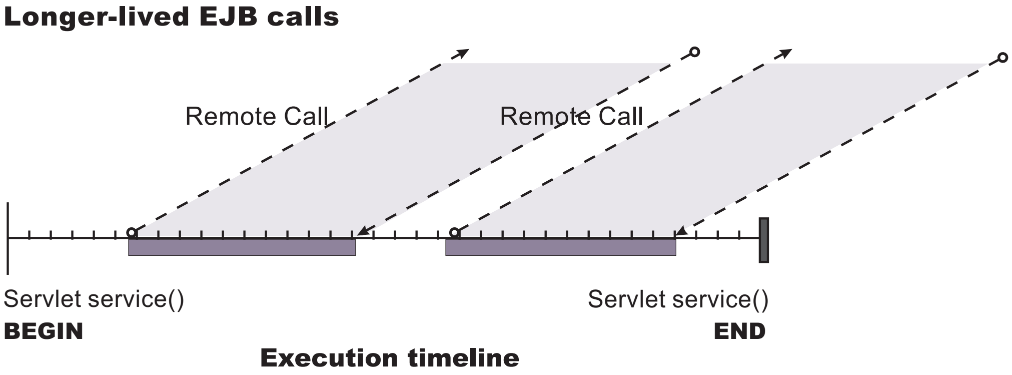


The following are two tips for queueing enterprise beans:

- **Analyze the calling patterns of the EJB client.**

  When configuring the thread pool, it is important to understand the calling patterns of the EJB client. If a servlet is making a small number of calls to remote enterprise beans and each method call is relatively quick, consider setting the number of threads in the ORB thread pool to a value lower than the Web container thread pool size value.





The degree to which the ORB thread pool value needs increasing is a function of the number of simultaneous servlets, that is, clients, calling enterprise beans and the duration of each method call. If the method calls are longer or the applications spend a lot of time in the ORB, consider making the ORB thread pool size equal to the Web container size. If the servlet makes only short-lived or quick calls to the ORB, servlets can potentially reuse the same ORB thread. In this case, the ORB thread pool can be small, perhaps even one-half of the thread pool size setting of the Web container.

- **Monitor the percentage of configured threads in use**.

  Tivoli Performance Viewer shows a metric called *percent maxed*, which is used to determine how often the configured threads are used. A value that is consistently in the double-digits, indicates a possible bottleneck a the ORB. Increase the number of threads.

See also ″Queuing network″ in the information center.

Using access intent policies

Apply access intent policies to methods of CMP entity beans.

Access intent policies

An access intent policy is a named set of properties (access intents) that governs data access for Enterprise JavaBeans (EJB) persistence. You can assign policies to an entity bean and to individual methods on an entity bean's home, remote, or local interfaces during assembly. You can set access intents only within EJB Version 2.x-compliant modules for entity beans with CMP Version 2.x.

This product supplies a number of access intent policies that specify permutations of read intent and concurrency control; the pessimistic/update policy can be qualified further. The selected policy determines the appropriate isolation level and locking strategy used by the run time environment.

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.

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.

Although it is recommended that you always configure bean level access intent for your applications, if you find it necessary you can apply access intent policies to methods within the scope of an EJB module. In such cases the policy becomes the default access intent for all requests upon the configured methods.

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.

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. See ″Assembling applications″ in the information center.

Note: 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 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

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. See ″Starting an assembly tool″ in the information center.
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 and processing) 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. In the Additional Properties area, select **Java Virtual Machine**.
8. In the Additional Properties area, select **Custom Properties**.
9. Select the **New** box.
10. In the **Name** entry field, type *com.ibm.websphere.ejbcontainer.initializeEJBsAtStartup*.
11. In the **Value** entry field, type *true*. Entering *true* causes all Enterprise JavaBeans to initialize when your application starts. Entering *false* causes initialization of all beans to be delayed.

    Note: Setting com.ibm.websphere.ejbcontainer.initializeEJBsAtStartup to either true or false takes precedence over any *Start EJB at Application Start* settings made on individual EJB types (see "Changing enterprise bean types to initialize at application start time using the Application Server Toolkit" on page 833).
12. 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. You can use the Tivoli Performance Viewer to view bean performance information.

To view this administrative console page, click **Servers > Application Servers >** *serverName* **> EJB Container Settings > EJB cache settings**.

Cleanup interval:

Specifies the interval at which the container attempts to remove unused items from the cache in order to reduce the total number of items to the value of the cache size.

The cache manager tries to maintain some unallocated entries that can be allocated quickly as needed. A background thread attempts to free some entries while maintaining some unallocated entries. If the thread runs while the application server is idle, when the application server needs to allocate new cache entries, it does not pay the performance cost of removing entries from the cache. In general, increase this parameter as the cache size increases.

Data type	Integer
Units	Milliseconds
Range	0 to 2 147 483 674
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

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. See ″Assembling EJB modules″, ″Assembling Web applications″, and ″Assembling applications″ in the information center.

1. Prepare the deployment environment.

2. Update the configuration for each EJB module as needed for the deployment environment. See ″Preparing to host applications″ in the information center.
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.

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.

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.
   b. Develop J2EE application client code.
   c. Develop pluggable application client code.
   d. Develop thin application client code.

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 ″Example: Developing an ActiveX application client to enterprise beans″. For more information about calling methods and access fields, see ″Example: Calling Java methods in the ActiveX to enterprise beans″ and ″Java field programming tips″ 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 Java primitive data type conversion values″ in the information center.

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 ″Threading tips″ for 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. See ″Assembly tools″ in the information center for additional information. 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. See ″Starting an assembly tool″ in the information center.
   - 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 (shown here on 2 lines for publication):

   ```
   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.

The following example illustrates the command line invocation syntax for the launchClient tool (shown here on 2 lines for publication):

```
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" in the *Troubleshooting and support* PDF.

-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. For more information on basic and advanced trace formatting, read the topic "Interpreting trace output" in the *Troubleshooting and support* PDF.

-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:

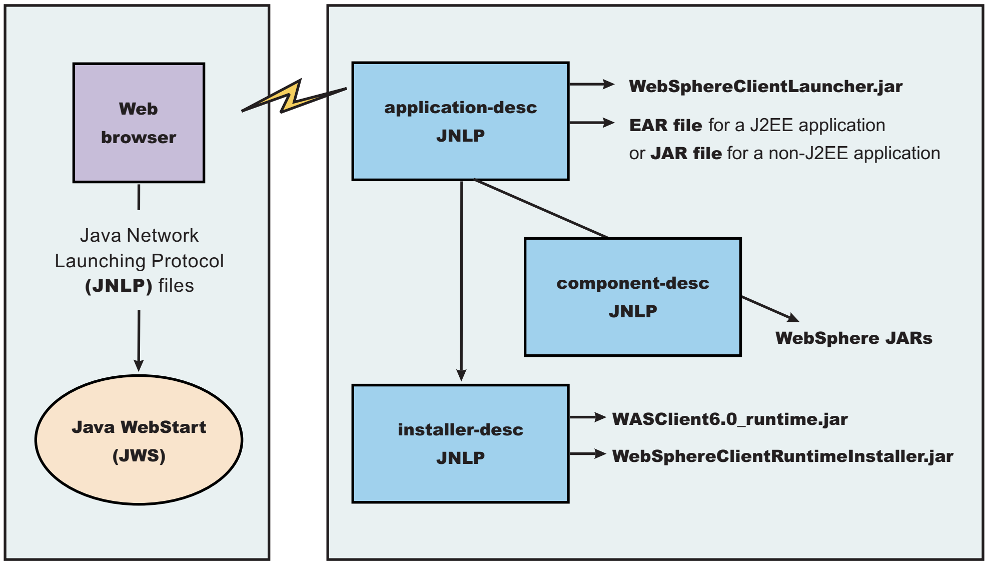
1. Specify a directory to extract the EAR file by setting the `com.ibm.websphere.client.applicationclient.archivedir` Java system property. If the directory does not exist or is empty, the EAR file is extracted normally. If the EAR file was previously extracted, the launchClient tool reuses the directory.
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.


Web browser
Java Network Launching Protocol (JNLP) files
Java WebStart (JWS)
application-desc JNLP
WebSphereClientLauncher.jar
EAR file for a J2EE application or JAR file for a non-J2EE application
component-desc JNLP
WebSphere JARs
installer-desc JNLP
WASClient6.0_runtime.jar
WebSphereClientRuntimeInstaller.jar

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 856
- “Client application Java Network Launcher Protocol deployment descriptor file”
- “ClientLauncher class” on page 861

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.

   Note: **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 (shown here on 2 lines for publication))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:

   ```
   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)
   ```

   Solution: Update the jnlp.jsp file to remove the quotation marks (″ ″) from the argument. Use the following example argument (shown here on 2 lines for publication) to correct the errors:

   ```
   <argument>-url=wsejb:/com.ibm.wssvt.tc.pli.ejb.WSMultiProtocolHome?jndiName=
     com/ibm/wssvt/tc/pli/ejb/WSMultiProtocolHome&</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 create a keystore file, such as `myKeystore`.
   b. You must also create a self-signed certificate for the `myKeystore` file. For more information, see the topic, ″Creating self-signed personal certificates″ in the *Securing applications and their environment* PDF.

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 create a keystore file, such as `myKeystore`.
   b. You must also create a self-signed certificate for the `myKeystore` file. For more information, see the topic, ″Creating self-signed personal certificates″ in the *Securing applications and their environment* PDF.
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 create a keystore file, such as `myKeystore`.
   b. You must also create a self-signed certificate for the `myKeystore` file. For more information, see the topic, ″Creating self-signed personal certificates″ in the *Securing applications and their environment* PDF.
7. Run buildClientLibJars to copy the Application Clients run-time library JAR files from the Application Client for WebSphere Application Server installation to a temporary directory. All the JAR files in the temporary directory are signed. See the following example for details:

```
buildClientLibJars C:\WebApp1\runtime\WebSphereJars
                   myKeystore myPassword myKeyAliasName
```

   a. This step also requires you to create a keystore file, such as `myKeystore`.
   b. You must also create a self-signed certificate for the `myKeystore` file. For more information, see the topic, ″Creating self-signed personal certificates″ in the *Securing applications and their environment* PDF.

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).

Installing Application Client for WebSphere Application Server

Running J2EE and Thin application clients that communicate with WebSphere Application Server requires that elements of the Application Server are installed on the machine on which the client runs. However, if the system does not have the Application Server installed, you can install Application Clients, which provide a stand-alone client run-time environment for your client applications. See the Supported Prerequisites page for more information on supported product platforms.

This article describes how to install the Application Client for WebSphere Application Server using the installation image on the product CD-ROM. The steps that follow provide enough detail to guide you through preparing for, choosing, and installing the variety of options and features provided. To prepare for installation and to make yourself familiar with installation options, complete the steps in this article and read the related topics, before you start to use the installation tools. Specifically, read these topics before installing the product:

- Installing silently
- Best practices for installing

As a general rule, if you launch an installation and there is a problem such as not having enough temporary space or not having the right packages on your Linux or UNIX systems, then cancel the installation, and make the required changes. Restart the installation to initiate your changes.

Although it is not supported or recommended, you can install this product as a non-root user on a UNIX operating system. However, you must use a user ID that is part of the administrator group on Windows platforms.

In Version 6, the Application Server for WebSphere Application Server is installable on a machine with a previous version of Application Clients. However, you cannot install a Version 6 Application Client on top of a previous version of the Application Client. For example, if a Version 5 Application Clients install under the `C:\WebSphere\AppClient` directory, you can not choose the same install location for your V6 Application Clients installation.

Note: For Application Client coexisting, there is a limitation on Applet client and ActiveX client on Windows that can not be coexisted with V5.0.x and V4.x of the clients. For example, the Applet client feature in Version 6 cannot coexist with the Applet client feature in any previous release. This coexistence is not available because the installation of Applet client feature in Version 6 sets the browser default Java Virtual Machine (JVM) using the Java Runtime Environment (JRE) from the Version 6 installation, which is Java Runtime Environment Version 1.4.2. Similarly, the ActiveX to EJB Bridge feature in Version 6 sets the Windows system path to use the JRE from the Version 6 installation.

1. Install Application Client for WebSphere Application Server using the launchpad. The launchpad program is available on the root directory of the product CD in the program, `launchpad.sh`, on Linux and UNIX platforms and `launchpad.bat` on Windows platforms.

   Note: The free download Application Client installation is not packaged with the launchPad program.

   a. Click **Launch the installation wizard for Application Client for WebSphere Application Server** from the launchpad tool to launch the InstallShield for MultiPlatforms installation wizard. This action launches the installation wizard.

      The Readme documentation to which the launchpad links is the `readme.html` file in the CD root directory. The readme directory off the root of the CD has more detailed Readme files. The Installation Guide is in the `/docs` directory of the CD root directory.

      Note: Readme file names are based on product offerings.

      When you install application clients, the current working directory must be the directory where the installer binary program is located. This placement is important because the resolution of the class files location is done in the current working directory. For example:

      ```
      cd /install_root/AppClient

      ./install
      ```

      or

      ```
       cd <CD mount point>/AppClient

           ./install
      ```

      Failing to use the correct working directory can cause ISMP errors that abort the installation.

      The installation wizard does not upgrade or remove previous Application Clients installation automatically. However, you must change the directory of any previously installed versions of the Application Client because you cannot install Version 6 on top of previously existing Application Client.

   b. As indicated in the previous example, you can start the installation wizard from the product CD-ROM, using the command line.

      On other Linux platforms and UNIX-based platforms, run the `./install` command.

      On Windows platforms, run the `Install.exe` command.

   c. You can also perform a silent installation using the `-options responsefile` parameter, which causes the installation wizard to read your responses from the options response file, instead of from the interactive graphical user interface. Customize the response file before installing silently. After customizing the file, issue the command to silently install. Silent installation is particularly useful if you install the product often.

The rest of this procedure assumes that you are using the installation wizard. There are corresponding entries in the response file for every prompt that is described as part of the wizard. Review the description of the response file for more information. Comments in the file describe how to customize its options.

2. Click **Next** to continue when the Welcome panel is displayed.
   a. Click the radio button beside the **I accept the terms in the license agreement** message if you agree to the license agreement, and click **Next** to continue.
3. Specify a destination directory. Click **Next** to continue.
   a. Ensure that there is adequate space available in the target directory.
   b. Specify a target directory for the Application Client product.
   c. Enter the required target directory to proceed to the next panel. Deleting the default target location and leaving an installation directory field empty prevents you from continuing the installation process.
4. Choose a type of installation, and click **Next**.

   If you use the GUI, you can choose a Typical installation type, which installs J2EE and Thin application client, Samples and IBM Developer Kit, Java 2 Technology Edition or a Custom installation type.

   The Custom installation type lets you select which features to install. However you can not install the J2EE and Java Thin application client feature and the Pluggable application client feature together.

   (Windows Only) If you select the **ActiveX to EJB Bridge** feature, then the following is displayed in a dialog box: `Do you want to add Java runtime to the system path and make it the default JRE?` If you answer **Yes**, then the Java run time is added to the beginning of the system path. If you answer **No**, then the ActiveX to EJB Bridge does not function from the Active Server Pages (ASP), unless you add the Java run time to the path. To add the Java run time later, see the topic ActiveX application clients or reinstall the Application Client.(Windows Only) If you select the **Applet client** feature, then the following message might be displayed: `An existing JDK or JRE has been detected on your computer. You chose to install the Applet Client, which will overwrite the registry entries for this JDK or JRE. Do you want to continue and install the Applet Client?` If you select **Yes**, the installation overrides the registry on your machine. If you select **No**, the Applet client feature is not installed, and you are directed back feature dialog box.
5. Install the Software Developer Kit, if you need to use any of the utilities that it provides, such as the javaccompiler, the jarutility or the jarsinger utility. The Java 2 Development Kit that IBM provides has two components, Java Runtime Environment (JRE) and a complete Software Developer Kit (SDK). The JRE sub feature is selected by default when the J2EE or Thin application client feature is selected. The SDK component is optional; however, you must install the SDK component to compile the sample.
6. Enter the host name of the WebSphere Application Server machine. Click **Next** to continue. The default port number for product Version 6 server is 2809.
7. Review the summary information, and click **Next** to install the product code or you might also click **Back** to change your specifications.
8. Click **Finish** to exit the wizard, after the Application Client installs.
9. Verify the success of the installer program by examining the Completion panel and the `<`*`install_root`*`>/logs/WAS.Client.isntall.log` file for installation status. The installer program records the following indicators of success in the logs:
   - `INSTCONFSUCCESS` indicates that the installation is successful and that no further log analysis is required.
   - `INSTCONFFAILED` indicates an installation failure that you cannot retry or recover from without reinstalling.

You successfully installed the Application Client for WebSphere Application Server and the features you selected.

If the installation is not successful, fix the error as indicated in the installation error message. For example, if you do not have enough disk space, add more space, and reinstall the Application Client.

Best practices for installing application clients

The following table offers tips for installing application clients on multiple platforms.

Operating environment	Tip
Linux and UNIX systems	Spaces are not supported in the name of the installation directory on Linux and UNIX platforms.
UNIX systems	When the application client installations are successful, the return code `1` is issued from the UNIX shell where you issued the `/install` command. Any other return code indicates an unsuccessful installation.
Solaris systems	Double-byte character set (DBCS) characters are not supported in the name of the installation directory on Solaris systems.
All platforms	Reserve at least 4 to 5MB free space in the target platform temporary directory.

All platforms	When specifying a different temporary directory while installing application clients, the following message is displayed if the target platform default temporary directory does not have enough free space to install application clients:
	`Error writing file =  There may not be enough`
	`temporary disk space.`
	`Try using -is:tempdir to use a temporary`
	`directory on a partition with more disk`
	`space.`
	Use the `-is:tempdir` installation option to specify a different temporary directory. For example, the following command uses `/swap` as a temporary directory during installation:
	`./install -is:tempdir /swap`

All platforms	After the installation, when changing the installation settings for the WebSphere Application Server host name and the port number, edit the `setupClient.bat` for Windows or `setupClient.sh` for UNIX. Change the `DEFAULTSERVERNAME` and `SERVERPORTNUMBER` to the new WebSphere Application Server host name and port number, respectively. If the `SERVERPORTNUMBER` is not set, then the default is 2809. Review the following example:
	`set DEFAULTSERVERNAME=NDServerName`
	`set SERVERPORTNUMBER=9810`
	The `setupClient.bat` file or `setupClient.sh` file is located in the `bin` sub-directory under the application clients installation destination.

Installing application clients silently

Use these steps to perform a silent installation, which uses the installation wizard to install the product. Instead of displaying a user interface, the silent installation provides interaction between you and the wizard by reading all of your responses from a file that you must customize.

1. Verify that the user ID that you are using to run the silent installation has sufficient authority to perform the task.
2. Customize the option response file.
   a. Locate the sample options response file. The file name is `setup.response` in the operating system platform directory on the product CD-ROM.
   b. Make a copy to preserve the original response file. For example, copy the file as `myoptionsfile`.
   c. Edit the copy in your flat file editor of choice, on the target operating system. Read the directions within the response file to choose appropriate values.

   Note: To prepare the file for a silent installation on AIX, use UNIX line-end characters (0x0D0A) to terminate each line of the options response file.

   d. Make a non-commented option to have a silent install.
   e. Include custom option responses that reflect parameters for your system.
   f. Follow the instructions in the response file to choose appropriate values.
   g. Save the file.

3. Issue a command to use your custom response file: `Install.exe -options myoptionsfile` for Windows platforms and `install -options ./myoptionsfile` for Linux and UNIX platforms.

   The sample options response file is located in the `operating-system platform` directory on the product CD-ROM.

   a. Issue the following command from a command prompt to update your response file: `-W silentInstallLicenseAcceptance.value=″true’`.

      Issuing this command changes the value for `silentInstallLicenseAcceptance.value` from `false` to `true`, which is necessary for installing application clients.

4. **Optional:** Restart your machine in response to the prompt that appears on Windows platforms when the installation is complete.

You installed application clients silently by using the response file.

To verify the silent install, look for the string `RC=INSTCONFSUCCESS` in the log file for successful installation and `RC=INSTCONFFAILED` for a failed install installation.

For UNIX platforms, the install command returns a return code of **1** to indicate a successful installation. Any other return code means that the installation failed.

Uninstalling Application Client for WebSphere Application Server

This task describes using the uninstaller program to uninstall the Application Client for WebSphere Application Server.

Before uninstalling, verify that you have no open Web browsers that are accessing the administrative console.

If you want to uninstall Application Clients manually, see the topic, ″Manually uninstalling on a Windows system″ in the information center.

1. Stop any browsers and any Java processes related to WebSphere Application Server products as described in ″Uninstalling the product″ in the information center.

2. Change directories to the `_uninst` directory before issuing the uninstall command. The command file is located in the *install_root*`/product/_uninst` directory on a Linux or UNIX platform, and in the *install_root*`\product\_uninst` directory on a Windows system.

   For example, to change directories before uninstalling the product from a Linux platform, issue this command if your installation root is `/opt/IBM/WebSphere/AppServer`:

   ```
   cd /opt/IBM/WebSphere/AppServer/product/_uninst
   ```

3. Issue the **uninstaller** command. The command file is named `uninstaller.bin` for Linux and UNIX platforms and `uninstaller.exe` on Windows platforms.

   Linux UNIX On Linux and UNIX platforms, issue the **uninstaller.bin** command from the *install_root*`/product/_uninst` directory:

   ```
   ./uninstaller.bin
   ```

   Windows On Windows platforms, call the **uninstaller.exe** command:

   ```
   install_root\product\_uninst\uninstaller.exe
   ```

Windows Call the program directly from the *install_root*\product_uninst directory. For example, if the installation root is C:\IBM\WebSphere\AppServer, issue the following command:

```
C:\IBM\WebSphere\AppServer\product\_uninst> uninstaller.exe
```

The Uninstaller wizard begins and displays the Welcome panel.

4. Click **Next** to begin uninstalling the product. The Uninstaller wizard displays a Confirmation panel that lists the product and features that you are uninstalling.
5. Click **Next** to continue uninstalling the product. The Uninstaller wizard deletes existing profiles first.

   After deleting profiles, the Uninstaller wizard deletes core product files by component.
6. Click **Finish** to close the wizard after the wizard removes the product.

Application Client for WebSphere Application Server is uninstalled.

Verify the uninstall procedure by viewing the *install_root*/product/logs/uninstlog.txt log file for errors. Look for the INSTCONFSUCCESS, indicating a successful uninstall in the log file:

```
Uninstall, com.ibm.ws.install.ni.ismp.actions.ISMPLogSuccessMessageAction, msg1,
   INSTCONFSUCCESS
```

Deploying J2EE application clients on workstation platforms

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

Related concepts

"Resource Adapters for the client" on page 874

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 (shown here on 2 lines for publication):

```
#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

Related tasks

"Configuring new JMS providers with the Application Client Resource Configuration Tool" on page 892

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. See ″Starting an assembly tool″ in the information center for additional information.
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.
2. (Optional) Migrate existing Web services.

   If you have used Web services based on Apache SOAP and now want to develop and implement Web services based on the Web Services for J2EE specification, you need to migrate client applications developed with all versions of 4.0, and versions of 5.0 prior to 5.0.2.
3. Deploying Web services

   This topic is a good starting point in learning about how to develop a J2EE Web service.
4. Configure Web services deployment descriptors

   You need to configure the deployment descriptors so that WebSphere Application Server can process the incoming Web services requests.
5. Assemble Web services.

   This topic presents what you need to assemble a Web service and in what order you should assemble the parts, for example an enterprise archive (EAR) file.
6. Deploy Web services.

   This topic presents the steps necessary to deploy the EAR file that has been configured and enabled for Web services.

7. Configure Web service client bindings. This topic explains how to edit bindings for a Web service after these bindings are deployed on a server. When one Web service communicates with another Web service, you must configure the client bindings to access the downstream Web service.
8. Publish the WSDL file. Refer to ″Publishing the WSDL file″ in the information center or in the *Administering applications and their environment* PDF.

   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.

   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.
11. Tune Web services.

    This topic includes information to help you use the Performance Monitoring Infrastructure (PMI) to measure the time required to process Web services requests.
12. Troubleshoot Web services.

    You can use this topic to learn more about troubleshooting different processes used to develop, implement and use Web services, including command-line tools, Java compiling errors, client runtime errors and exceptions, serialization and deserialization errors, and authentication challenges and authorization failures with Web services security.

The following example illustrates how a business might use Web services.

The owner of a flower shop wants to start receiving orders from customers through the Web. This owner starts the process by finding wholesale flower suppliers, pricing the product, and completing contracts for future flower orders.

Using Web services, the flower shop owner can find wholesale flower suppliers.

The flower shop owner can request price lists from each of the suppliers by obtaining a 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 ″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 they incorporated the Web services concept.

Web services

Web services are self-contained, modular applications that you can describe, publish, locate, and invoke over a network.

WebSphere Application Server supports Web services that are developed and implemented based on the Web Services for Java 2 Platform, Enterprise Edition (J2EE) specification.

A typical Web services scenario is a business application requesting a service from another existing application. The request is processed through a given Web address using SOAP messages over a HTTP, Java Message Service (JMS) transport or invoked directly as Enterprise JavaBeans (EJB). The service receives the request, processes it, and returns a response. Examples of a simple Web service include weather reports or getting stock quotes. The method call is synchronous, that is, it waits until the result is available. Transaction Web services, supporting quotes, business-to-business (B2B) or business-to-client (B2C) operations include airline reservations and purchase orders.

Web services can include the actual service or the client that accesses the service.

Web services are Web applications that help you be more flexible in your business processes by integrating with applications that otherwise do not communicate. The inner-library loan program at your local library is a good example of the Web services concept and its evolution. The Web service concept existed even before the term; the concept became widely accepted with the creation of the Internet. Before the Internet was created, you visited your library, searched the collections and checked out your books. If you did not find the book that you wanted, the librarian did a search for you by computer or phone and located the book at a nearby library. The librarian ordered the book for you and you picked it up after it was delivered to your local library. By incorporating Web services applications, you can streamline your library visit.

Now, you can search the local library collection and other local libraries at the same time. When other libraries provide your library with a Web service to search their collection (the service might have been provided through UDDI), your results yield their resources. Another Web service application might enable you to check the book out and get it sent to your home. Using Web services applications saves time and provides a convenience for you, as well as freeing the librarian to do other business tasks.

Web services reflect the service-oriented architecture (SOA) approach to programming. This approach is based on the idea of building applications by discovering and implementing network-available services, or by invoking the available applications to accomplish a task. Web services deliver interoperability, for example, Web services applications provide components created in different programming languages to work together as if they were created using the same language. Web services rely on existing transport technologies, such as HTTP, and standard data encoding techniques, such as Extensible Markup Language (XML), for invoking the implementation.

The key components of Web services include:

- Web Services Description Language (WSDL)

WSDL is the XML-based file that describes the Web service. The Web service request uses this file to bind to the service.

- SOAP

  SOAP is the XML-based protocol that the Web service request uses to invoke the service.

For a more detailed scenario, see ″Web services scenario: Overview″ 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 modify your encoding from UTF-8 to UTF-16 if you need to change from UTF-8 to UTF-16.

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

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

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

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

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

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

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 WSDL EJB bindings to invoke an EJB from a Web services client ″ in the information center.

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: 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.
- 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.

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.

Protect your Web services applications by adding security to the applications.

wsdeploy command

This topic explains how to use the **wsdeploy** command-line tool with Web services that are based on the Web Services for Java 2 Platform, Enterprise Edition (J2EE) specification. The **wsdeploy** command adds WebSphere product-specific deployment classes to a Web services-compatible enterprise application enterprise archive (EAR) file or an application client Java archive (JAR) file. These classes include:

- Stubs

- Serializers and deserializers
- Implementations of service interfaces

This deployment step must be performed at least once, and can be performed more often. Deployment can be performed separately using the **wsdeploy** command, assembly tools, or when the application is installed. When using the **wsadmin** command for installation, specify the **-deployws** option.

The **wsdeploy** command operates as noted in the following list:
- Each module in the enterprise application or JAR file is examined
- If the module contains Web services implementations, indicated by the presence of the `webservices.xml` deployment descriptor, the associated Web Services Description Language (WSDL) files are located and the **WSDL2Java** command is run with the role deploy-server option.
- If the module contains Web services clients, indicated by the presence of the client deployment descriptor, the associated WSDL files are located and the **WSDL2Java** command is run with the role deploy-client option.
- The files generated by the **WSDL2Java** command are compiled and repackaged.

See **WSDL2Java** command for more information about the files that are generated for deployment.

When the generated files are compiled, they can reference application-specific classes outside the EAR or JAR file, if the EAR or JAR file is not self-contained. In this case, use either the -jardir or -cp option to specify additional JAR or zip files to be added to CLASSPATH variable when the generated files are compiled.

wsdeploy command syntax

The command syntax is noted in the following example:

```
wsdeploy Input_filename Output_filename [options]
```

Required options:
- ***Input_filename***

  Specifies the path to the EAR or JAR file to deploy.
- ***Output_filename***

  Specifies the path of the deployed EAR or JAR file. If *output_filename* already exists, it is silently overwritten. The *output_filename* can be the same as the*input_filename*.

Other options:
- **-jardir** *directory*

  Specifies a directory that contains JAR or zip files. All JAR and zip files in this directory are added to the CLASSPATH used to compile the generated files. This option can be specified zero or more times.
- -**cp** *entries*

  Specifies entries to add to the CLASSPATH when the generated classes are compiled. Multiple entries are separated the same as they are in the CLASSPATH environment variable, with a semicolon on Windows platforms and a colon for Linux and Unix platforms.
- **-codegen**

  Specifies to generate but not compile deployment code. This option implicitly specifies the -keep option.
- **-debug**

  Includes debugging information when compiling, that is, use javac -g to compile.
- **-help**

  Displays a help message and exit.
- **-ignoreerrors**

  Do not stop deployment if validation or compilation errors are encountered.
- **-keep**

Do not delete working directories containing generated classes. A message is displayed indicating the name of the working directory that is retained.

- **-novalidate**

  Do not validate the Web services deployment descriptors in the input file.
- **-trace**

  Displays processing information, including the names of the generated files.

Example The following example illustrates how the options are used with the **wsdeploy** command:

```
wsdeploy x.ear x_deployed.ear -trace -keep
Processing web service module x_client.jar.
Keeping directory: f:\temp\Base53383.tmp for module: x_client.jar.
Parsing XML file:f:\temp\Base53383.tmp\WarDeploy.wsdl
Generating f:\temp\Base53383.tmp\generatedSource\com\test\WarDeploy.java
Generating f:\temp\Base53383.tmp\generatedSource\com\test\WarDeployLocator.java
Generating f:\temp\Base53383.tmp\generatedSource\com\test\HelloWsBindingStub.java
Compiling f:\temp\Base53383.tmp\generatedSource\com\test\WarDeploy.java.
Compiling f:\temp\Base53383.tmp\generatedSource\com\test\WarDeployLocator.java.
Compiling f:\temp\Base53383.tmp\generatedSource\com\test\HelloWsBindingStub.java.
Done processing module x_client.jar.
```

Messages

- Flag *-f*is not valid.

  Option *f* was not recognized as a valid option.
- Flag *-c* is ambiguous.

  Options can be abbreviated, but the abbreviation must be unique. In this case, the **wsdeploy** command cannot determine which option was intended.
- Flag *-c* is missing parameter *-p*.

  A required parameter for an option is omitted.
- Missing *p* parameter.

  A required option is omitted.

Configuring Web service client bindings

When a Web service application is deployed into WebSphere Application Server, an instance is created for each application or module. The instance contains deployment information for the Web module or enterprise JavaBean (EJB) module, including client bindings.

Deploy the Web service into WebSphere Application Server.

To complete this task, you need to know the topology of the URL endpoint address of the Web services servers and which Web service the client depends upon. You can view the deployment descriptors in the administrative console to find topology information. See the article ″View Web services server deployment descriptors″ in the information center for more information.

The client bindings define the Web Services Description Language (WSDL) file name and preferred ports. The relative path of a Web service in a module is specified within a compatible WSDL file that contains the actual URL to be used for requests. The address is only needed if the original WSDL file did not contain a URL, or when a different address is needed. For a service endpoint with multiple ports, you need to define an alternative WSDL file name.

The following steps describe how to edit bindings for a Web service after these bindings are deployed on a server. When one Web service communicates with another Web service, you must configure the client bindings to access the downstream Web service.

You can also configure client bindings with **wsadmin**.

To configure client bindings through the administrative console:

1. Open the administrative console.
2. Click **Applications** > **Enterprise Applications** > *application_instance* > **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.
   1. 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.
   2. 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.
   1. 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 milliseconds, 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.

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 in the information center:
- 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.

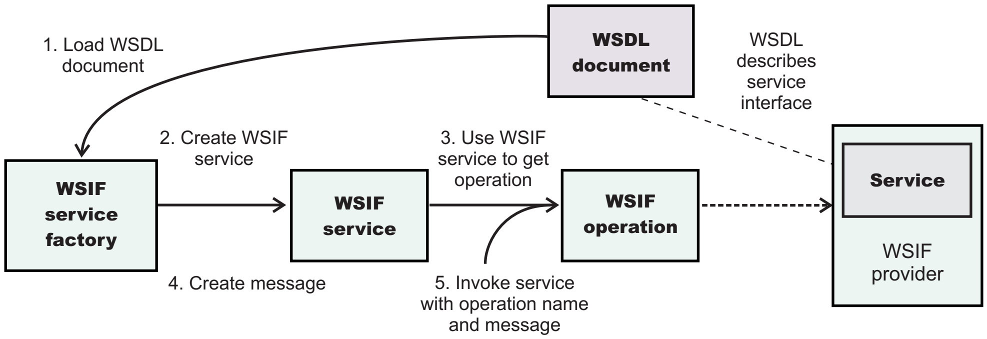
WSIF provides the following features:
- An API that provides binding-independent access to any Web service.
- A close relationship with WSDL, so it can invoke any service that you can describe in WSDL.
- A stubless and completely dynamic invocation of a Web service.
- The capability to plug a new or 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. You can use the Java Naming and Directory Interface (JNDI) to find the WSIF service, or you can use the location described in the WSDL.

For more conceptual information about WSIF and WSDL, see the following topics:
- WSIF and WSDL
- WSIF architecture
- 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 ″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.

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.

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″ in the information center 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″ in the information center 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″ in the information center 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″ in the information center 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″ in the information center introduces some terms with which you might need to be familiar with to use a UDDI Registry.

″UDDI Registry user interface″ in the information center 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″ in the information center 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 Programmingexplains 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″ in the information center 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″ in the information center might be useful if you encounter any problems or unexpected behaviour while using the UDDI Registry, and ″UDDI Registry messages″ in the information center 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″ 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 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 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.
6. Review Web services Samples.

You have a design plan for implementing Web services applications into your business architecture.

Develop a Web service.

This topic explains how to develop a Web service using the Web Services for J2EE specification.

Service-oriented architecture

A *service-oriented architecture (SOA)* is a collection of services that communicate with each other, for example, passing data from one service to another or coordinating an activity between one or more services.

Companies want to integrate existing systems to implement Information Technology (IT) support for business processes that cover the entire business value chain. A variety of designs are used, ranging from rigid point-to-point electronic data interchange (EDI) to Web auctions. By using the Internet, companies 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:

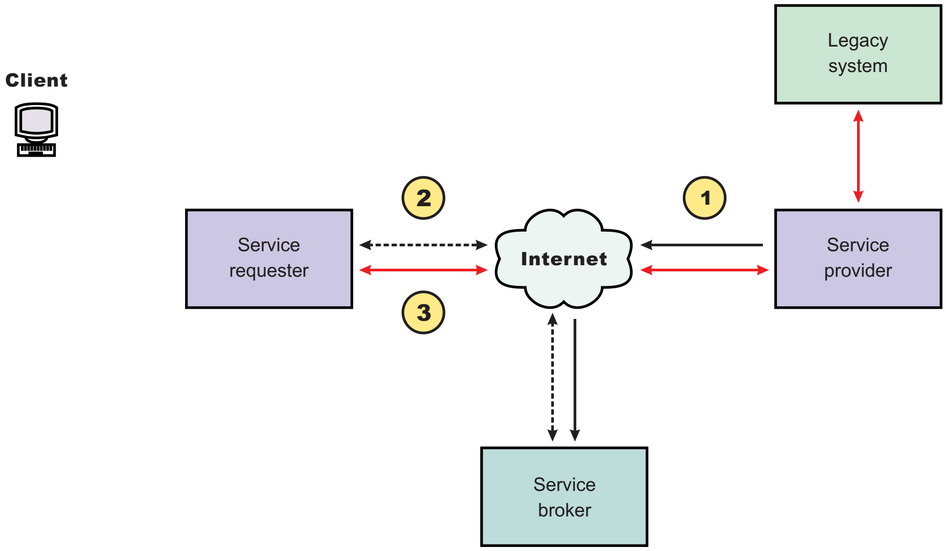
- **Service provider**

  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 Version 6 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 can not be changed for this configuration.

Important: Important Note for DB2 users - Starting the WebSphere Application Server in which the UDDI Registry is deployed

On Unix and Linux platforms, before you start the WebSphere server that is hosting your UDDI DB2 application, you must run the db2profile script which is found in the home directory for you db2 userid, for example /home/db2inst1/sqllib/db2profile

Similarly when in an ND configuration and starting the server via the Deployment Manager, you must ensure the db2profile is run before starting the relevant nodeagent.

The information above is true during initial set up and deployment of UDDI and also every time you start the application server or node agent. For this reason it is strongly recommended that you update the user profile for the user that will start the nodeagent and server to also execute the db2profile.

If running the profile manually type:

```
. /home/db2inst1/sqllib/db2profile
```

In the above example, notice that the ’.’ is followed by a single space character.

Deploying into a Network Deployment cell (but not a cluster)

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

It is important to note that uddiDeploy.jacl and uddiRemove.jacl can not 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 the addNode command for details. This applies to a default or a user 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 (both policies, properties and user data) will be lost in the move.

Moving between Cloudscape, DB2 and Oracle 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 a default UDDI node, or a user customized UDDI node.

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:
   - Creating a DB2 database
   - Creating an Oracle database
   - Creating a Cloudscape database

Note: You must not run the final step to set the default node indicator in the database.

2. Create a J2C Authentication Data Entry (not required for Cloudscape):
   a. Expand **Global Security** and **JAAS Configuration** (on the right), then click on **J2C Authentication Data**
   b. Select **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, which is used to read and write to the UDDI registry database.

      Password
      The password associated with the userid specified above.

      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):
   - For DB2, select the options **DB2 Legacy CLI-based Type 2 JDBC Driver** and **Connection Pool datasource**.
   - For Oracle, select the options **Oracle JDBC Driver** and **Connection Pool datasource**.
   - For Cloudscape, select the **Cloudscape JDBC Driver** and **Connection Pool datasource**.

   For details on how to create a JDBC provider, see Creating and configuring a JDBC provider using the administrative console.
4. Create a datasource for the UDDI Registry by following these steps:
   a. Expand **Resource** and **JDBC Providers**
   b. Select the desired 'scope' of the JDBC provider you selected or created earlier. For example, select:

      ```
      Server: yourservername
      ```

      This displays 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. Select '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.

      Description
      a suitable description

      Category
      set to uddi

      Data store helper class name
      filled in for you as:

- for **DB2** - com.ibm.websphere.rsadapter.DB2DataStoreHelper
- for **Oracle 9i** - com.ibm.websphere.rsadapter.OracleDataStoreHelper
- for **Oracle 10g** - com.ibm.websphere.rsadapter.Oracle10gDataStoreHelper
- for **Cloudscape** - com.ibm.websphere.rsadapter.CloudscapeDataStoreHelper

Relational Database Management System **data source properties**

- for **DB2** - **Database name** - for example:

  ```
  UDDI30
  ```

- for **Oracle** - **URL** - for example:

  ```
   jdbc:oracle:oci8:@<Oracle database name>
  ```

- for **Cloudscape** - **Database name** - for example:

  ```
  ${USER_INSTALL_ROOT}/databases/com.ibm.uddi/UDDI30
  ```

Component-managed authentication alias

- for **DB2**, select the alias that you created in step 3 from the pulldown. It will have the node name appended in front of it, for example MyNode/UDDIAlias.
- for **Oracle**, select the alias that you created in step 3 from the pulldown. It will have the node name appended in front of it, for example MyNode/UDDIAlias.
- for **Cloudscape** leave this set to (none).

Container-managed authentication alias
Set to DefaultPrincipalMapping

Mapping-configuration alias
Set to (none)

g. Use this Data Source in container-managed persistence (CMP), and ensure it is unchecked.

5. Test the connection to your UDDI database by clicking on the ″Test Connection″ button for the datasource. 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 into the server by running the uddiDeploy.jacl command from a command prompt as shown.

   This file is located in

   ```
   WAS_HOME/bin
   ```

   The syntax of the command is:

   ```
   wsadmin [-conntype none] -f uddiDeploy.jacl <node> <server>
   ```

   where:

   -conntype none
   is optional, and is only needed if the application server is not running.

   <node>
   the name of the WebSphere node on which the target server runs. Note that the node name is case sensitive.

   <server>
   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.

   You are recommended to deploy the UDDI application using the uddiDeploy.jacl, 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 command in a command prompt will do this for you.

For example, to deploy UDDI on node ’MyNode’ and server ’server1’ on a Windows system you might enter the following (assuming that server1 is already started):

```
wsadmin -f uddiDeploy.jacl MyNode server1
```

You should now start the UDDI application, or start the application server if it is not already running.

As you have chosen a user customized UDDI node, you will need to set up the properties for the UDDI node using UDDI administration, and initialize the node before it is ready to accept UDDI requests (see ″Configuring the UDDI Registry Application″ in the information center for details).

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 a 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 configuration Network Deployment read the section ’Installing into a Network Deployment Cell’ first.

   Deploy the UDDI Registry by running the uddiDeploy.jacl script from a command prompt as shown.

   This file is located in

   ```
   WAS_HOME/bin
   ```

   The syntax of the command is (shown here on 2 lines for publication):

```
wsadmin [-conntype none] -wsadmin_classpath WAS_HOME/cloudscape/lib -f
   uddiDeploy.jacl <node> <server> default
```

   where:

   -conntype none
   is optional, and is only needed if the application server is not running.

   WAS_HOME
   is the directory name of the WebSphere Application Server install location

   <node>
   is the name of the WebSphere node on which the target server runs. Note that the node name is case sensitive.

   <server>
   is the name of the target server on which you wish to deploy the UDDI Registry, such as server1. Note the server name entered is case sensitive.

   default
   using the ’default’ option 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.

   For example, to create a UDDI node called ’MyNode’ on server ’server1’ on a Windows system, you might enter the following (this assumes server1 is started). The command is shown on 2 lines for publication.

```
wsadmin -wsadmin_classpath C:\Progra~1\IBM\WebSphere\AppServer\cloudscape\lib
   -f uddiDeploy.jacl MyNode server1 default
```

   If the server is not started the command is (shown here on 2 lines for publication):

```
wsadmin -conntype none -wsadmin_classpath C:\Progra~1\IBM\WebSphere\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.

2. **Optional: For a default DB2, default Cloudscape or default Oracle UDDI node:**

   DB2, Cloudscape or Oracle may be used for a single application server installation or a Network Deployment installation.

   a. Initially you need to define some resources and run some scripts to set up the database. Go to Creating the DB2 database, Creating the Cloudscape databaseor Creating the Oracle database to create and load the database, ensuring you have run the final step to set the default node indicator in the database, and return to this point. You must create a JDBC Provider if a suitable one does not already exist or select one, and a Datasource to reference the UDDI database.

      - For DB2, select the 'DB2 Legacy CLI-based Type 2 JDBC Driver' option and Connection Pool datasource option.
      - When using Oracle ensure you select 'Oracle JDBC Driver' and the Connection Pool datasource option.
      - For Cloudscape, select the supplied **Cloudscape JDBC Driver**
      - Refer to Creating and configuring a JDBC provider using the administrative console for details on how to create a JDBC Provider.

   b. Create a J2C Authentication Data Entry for DB2 or Oracle access (not required for Cloudscape) using the WebSphere Application Server administrative console by following these steps:

      1) Expand 'Global Security' and 'JAAS Configuration' and click on 'J2C Authentication Data Entries'
      2) Select '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
         your DB2 userid, such as db2admin, or your Oracle userid, such as SYSTEM.

         Password
         The password associated with your userid.

         Description
         a suitable description

      Click 'Apply' and then Save the changes to the master configuration

   c. Create a datasource for the UDDI Registry by following these steps:

      1) Expand 'Resource' 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 your DB2 or Oracle JDBC provider as selected or created earlier
      4) Under 'Additional Properties', select 'Data Sources' (**not** the Data Sources Version 4 option)
      5) Select '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: Note that 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.

Description
a suitable description

Category
set to uddi

Data store helper class name
filled in for you as:

- for **DB2** - com.ibm.websphere.rsadapter.DB2DataStoreHelper
- for **Oracle 9i** - com.ibm.websphere.rsadapter.OracleDataStoreHelper
- for **Oracle 10g** - com.ibm.websphere.rsadapter.Oracle10gDataStoreHelper
- for **Cloudscape** - com.ibm.websphere.rsadapter.CloudscapeDataStoreHelper

Relational Database Management System **data source properties**

- for **DB2** - **Database name** - for example:
  ```
  UDDI30
  ```
- for **Oracle** - **URL** - for example:
  ```
   jdbc:oracle:oci8:@<Oracle database name>
  ```
- for **Cloudscape** - **Database name** - for example:
  ```
  ${USER_INSTALL_ROOT}/databases/com.ibm.uddi/UDDI30
  ```

Component-managed authentication alias

- for **DB2**, select the alias that you created in step 3 from the pulldown. It will have the node name appended in front of it, for example MyNode/UDDIAlias.
- for **Oracle**, select the alias that you created in step 3 from the pulldown. It will have the node name appended in front of it, for example MyNode/UDDIAlias.
- for **Cloudscape** leave this set to (none).

Container-managed authentication alias
select the alias that you created earlier from the pulldown. It will have the node name appended in front of it, for example MyNode/UDDIAlias.

Mapping-configuration alias
set to DefaultPrincipleMapping from the pulldown (this might already be filled in).

Leave all other fields unchanged.

Click ’Apply’ and Save the changes to the master configuration.

d. Test the connection to your UDDI database by clicking on the ″Test Connection″ button for the datasource. 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.

e. Deploy the UDDI Registry into the server by running the uddiDeploy.jacl script from a command prompt as shown.

This file is located in

```
WAS_HOME/bin
```

The syntax of the command is:

```
wsadmin [-conntype none] -f uddiDeploy.jacl <node> <server>
```

where:

-conntype none
is optional, and is only needed if the application server is not running.

<node>
the name of the WebSphere node on which the target server runs. Note that the node name is case sensitive.

<server>
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.

You are recommended to deploy the UDDI application using the uddiDeploy.jacl, 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 command in a command prompt will do this for you.

For example, to deploy UDDI on node 'MyNode' and server 'server1' on a Windows system you might enter the following (assuming that server1 is already started):

```
wsadmin -f uddiDeploy.jacl MyNode server1
```

You should now start the UDDI application, or start the application server if it is not already running.

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

Perform this task if you want to use DB2 as the database store for your UDDI Registry data. You only need do this once for each UDDI Registry, as part of Setting up and deploying a UDDI Registry.

Before you begin

The steps below use a number of variables for which you need to enter appropriate values. You should decide the values that you will use before you start. The variables used, and suggested values are:

<DataBaseName>
is the name of the UDDI Registry database. A recommended value is UDDI30, and this name is assumed throughout the UDDI documentation. If you use some other name, then you should substitute that name whenever you see 'UDDI30' in other sections of the documentation.

<DB2UserID>
is a DB2 userid with administrative privileges.

<DB2Password>
is the password for the DB2 userid.

<BufferPoolName>
is the name of a buffer pool to be used by the UDDI Registry database. A suggested name is uddibp, but any name can be used, as the buffer pool is created as part of this task.

<TableSpaceName>
is the name of a table space. A suggested value is uddits, but any name can be used.

<TempTableSpaceName>
is the name of a temporary table space. A suggested value is udditstemp, but any name can be used, as the temporary table space is created as part of this task.

To create the DB2 database follow the steps shown below:

1. Run the following commands to setup the DB2 environment variables from the DB2 Command Line Processor (that is, at a prompt which requires db2 to be entered before each command):
   a. Not required for DB2 Vesion 8 and onwards: `db2set DB2_INDEX_2BYTEVARLEN=YES`
   b. `db2set DB2CODEPAGE=1208`
2. Create the DB2 database by entering the following command from the DB2 Command Line Processor:

a. `db2 create database <DataBaseName> using codeset UTF-8 territory en` where <DataBaseName> is the name of the database being created.

3. Configure the DB2 database by entering the following commands from the DB2 Command Line Processor:
   a. `db2 connect to <DataBaseName> user <DB2UserID> using <DB2Password>`
   b. `db2 update db cfg for <DataBaseName> using applheapsz 2048`
   c. `db2 update db cfg for <DataBaseName> using logfilsiz 8192`
   d. `db2 connect reset`
   e. `db2 terminate`
4. Create additional database structures by entering the following commands from the DB2 Command Line Processor:
   a. `db2 connect to <DataBaseName> user <DB2UserID> using <DB2Password>`
   b. `db2 create bufferpool <BufferPoolName> size 250 pagesize 32K`
   c. `db2 connect reset`
   d. `db2 terminate`
   e. `db2 force application all`
   f. `db2 terminate`
   g. `db2stop`
   h. `db2start`
5. Create further database structures by entering the following commands from the DB2 Command Line Processor:
   a. `db2 connect to <DataBaseName> user <DB2UserID> using <DB2Password>`
   b. `db2 create regular tablespace uddits pagesize 32K managed by system using ('<TableSpaceName>') extentsize 64 prefetchsize 32 bufferpool <BufferPoolName>`
   c. `db2 create system temporary tablespace <TempTableSpacename> pagesize 32K managed by system using ('<TempTableSpacename>') extentsize 32 overhead 14.06 prefetchsize 32 transferrate 0.33 bufferpool <BufferPoolName>`
6. Enter the following commands exactly as shown (noting in particular one step uses -vf rather than -tvf) into a DB2 Command Window to define the database structures needed to store the UDDI data (before running these commands change dirstory to WAS_HOME/UDDIReg/databaseScripts):
   a. `db2 -tvf uddi30crt_10_prereq_db2.sql`
   b. `db2 -tvf uddi30crt_20_tables_generic.sql`
   c. `db2 -tvf uddi30crt_25_tables_db2udb.sql`
   d. `db2 -tvf uddi30crt_30_constraints_generic.sql`
   e. `db2 -tvf uddi30crt_35_constraints_db2udb.sql`
   f. `db2 -tvf uddi30crt_40_views_generic.sql`
   g. `db2 -tvf uddi30crt_45_views_db2udb.sql`
   h. `db2 -vf uddi30crt_50_triggers_db2udb.sql`
   i. `db2 -tvf uddi30crt_60_insert_initial_static_data.sql`
7. This last step should only be run if you want your database to be used as a default UDDI node.
   a. Enter the following command from the DB2 Command Line Window: `db2 -tvf uddi30crt_70_insert_default_database_indicator.sql`

Continue with setting up and deploying your UDDI Registry node.

Creating an Oracle database

Perform this task if you want to use Oracle as the database store for your UDDI Registry data. You need only do this once for each UDDI registry, as part of Setting up and deploying a UDDI Registry.

Note: Only Version 9i[1] and Oracle 10g[2]

Before you begin

Note that this task will create two new schemas, ibmudi30 and ibmuds30. You will be unable to complete this task if you already have existing schemas of those names.

The steps below use a number of variables for which you need to enter appropriate values. You should decide the values that you will use before you start. The variables used, and suggested values are:

<OracleUserID>
is the Oracle userid to be used to create the database.

<OraclePassword>
is the password for the Oracle userid.

1. Run the following commands:
   a. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_10_prereq_oracle.sql`
   b. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_20_tables_generic.sql`
   c. Complete one of the two following actions depending on your level of Oracle:
      - For Version 10g and later:
        ```
        sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_25_tables_oracle.sql
        ```
      - For Oracle 9i:
        ```
        sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_25_tables_oracle_pre10g.sql
        ```
   d. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_30_contraints_generic.sql`
   e. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_35_constraints_oracle.sql`
   f. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_40_views_generic.sql`
   g. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_45_views_oracle.sql`
   h. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_50_triggers_oracle.sql`
   i. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_60_insert_initial_static_data.sql`
2. This last command should only be run if you want the database to be used as a default UDDI node.
   ```
   sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_70_insert_default_database_indicator.sql
   ```

Continue with setting up and deploying your UDDI Registry node.

Creating a Cloudscape database

Perform this task if you want to use Cloudscape 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

[1]

Restrictions:
discoveryURL (Business) maximum 4000 bytes, UDDI specification 4096 characters; accessPoint (bindingTemplate) maximum 4000 bytes), UDDI Specification 4096 characters; instacneParms (tModelInstanceInfo) maximum 4000 bytes, UDDI specification 8192 characters; overviewURL (tModelInstanceInfo) maximum 4000 bytes, UDDI specification 4096 characters; Digital Signature maximum 4000 bytes.

[2]

Restrictions:
discoveryURL (business) maximum 4000 bytes, UDDI specification 4096 characters.

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 WAS_HOME/UDDIReg/databasescripts

arg2 is the path to the location where you want to install the Cloudscape database, such as WAS_HOME/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 some other 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 to create a UDDI Cloudscape database using UDDICloudscapeCreate.jar, which is created in WAS_HOME/Lib directory.

1. On Solaris and HP-UX platforms:

   ```
   java -Djava.ext.dirs=WAS_HOME/cloudscape/lib:WAS_HOME/java/lib/ext -jar
             UDDICloudscapeCreate.jar arg1 arg2 arg3 arg4
   ```

2. On all other platforms:

   ```
   java -cp WAS_HOME/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.

Remote Oracle

Create the Oracle tables used for an Oracle UDDI database on the remote system, and use the URL property of the UDDI datasource to reference the remote Oracle instance.

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.

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.

Open a browser window and enter the URL to access the UDDI Registry User Interface. Under the ’Quick Find’ heading on the Find tab, select the ’Business’ radio button and enter ’%’ in the box. Click the ’Find’ button, and you should see the business displayed in the detail frame which will be this UDDI node’s business entity. You can click on the listed business in order to see its detail.

If you see a business listed in the detail frame, your UDDI Registry has been deployed successfully.

As a further IVP, 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.
   - Configure the URL endpoint information for JMS bindings.
   - Configure the URL endpoint information to directly access enterprise beans.
2. Externalize or publish the WSDL file out of the application. You can complete this task in one of three ways:
   - Publish a WSDL file with the administrative console
   - Publish a WSDL file through a URL.
   - Publish a WSDL file with the **wsadmin** command tool.

Apply security to the Web service.

WSDL

Web Services Description Language (WSDL) is an Extensible Markup Language (XML)-based description language. This language was submitted to the World-Wide Web Consortium (W3C) as the industry standard for describing Web services. The power of WSDL is derived from two main architectural principles: the ability to describe a set of business operations and the ability to separate the description into two basic units. These units are a description of the operations and the details of how the operation and the information associated with it are packaged.

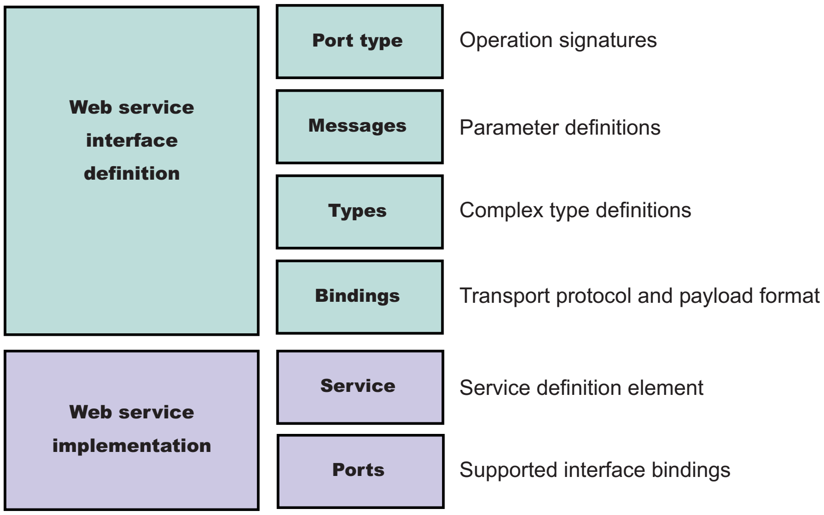
A WSDL document defines services as collections of network endpoints, or ports. In WSDL, the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. This separation supports the reuse of abstract definitions: messages, which are abstract descriptions of exchanged data, and port types, which are abstract collections of operations. The concrete protocol and data format specifications for a particular port type constitutes a reusable binding. A port is defined by associating a network address with a reusable binding, and a collection of ports define a service. Therefore, a WSDL document is composed of several elements. See WSDL architecture for more information and examples of the WSDL elements.

When creating Web services for WebSphere Application Server, you must first have an implementation bean that includes a service endpoint interface. Then, you use the **Java2WSDL** command-line tool to create a WSDL file that defines the Web services. To learn more about how the WSDL file is used in the development process, see Developing Web services.

WSDL architecture

Web Services Description Language (WSDL) files are written in Extensible Markup Language (XML). To learn more about XML, see Web services: Resources for learning.

The following is the structure of the information in a WSDL file:


Web service interface definition
Port type
Operation signatures
Messages
Parameter definitions
Types
Complex type definitions
Bindings
Transport protocol and payload format
Web service implementation
Service
Service definition element
Ports
Supported interface bindings

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.

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 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.
2. Follow the steps in one of the following topics, depending on what task you want to complete:
   - Configure Web service client bindings.
   - Configure Web service client preferred port mappings .
   - Configure Web service client port information .
   - Configure the scope of a Web service port .

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.
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
wsif.asyncrequest.timeout=60
```

To enable your legacy Web services to continue to work with WSIF, you might need to change the default WSIF SOAP provider back to the former Apache SOAP provider.

Enabling security for WSIF

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 the *Troubleshooting and support* PDF.

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 Rational Application Developer 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 Rational Application Developer 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 Address Book Sample.

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.
- **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.

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 ″Enabling trace″ and ″Setting up component trace (CTRACE)″ in the *Troubleshooting and support* PDF.

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.

For more information about the message identifier format, see the topic Message reference.

WSIF0001E: An extension registry was not found for the element type “{0}”

Explanation: Parameters: {0} element type. No extension registry was found for the element type specified.

User Response: Add the appropriate extension registry to the port factory in your code.

WSIF0002E: A failure occurred in loading WSDL from “{0}”

Explanation: Parameters: {0} location of the WSDL file. The WSDL file could not be found at the location specified or did not parse correctly

User Response: Check that the location of the WSDL file is correct. Check that any network connections required are available. Check that the WSDL file contains valid WSDL.

WSIF0003W: An error occurred finding pluggable providers: {0}

Explanation: Parameters: {0} specific details about the error. There was a problem locating a WSIF pluggable provider using the J2SE 1.3 JAR file extensions to support service providers architecture. The WSIF trace file will contain the full exception details.

User Response: Verify that a META-INF/services/org.apache.wsif.spi.WSIFProvider file exists in a provider jar, that each class referenced in the META-INF file exists in the class path, and that each class implements org.apache.wsif.spi.WSIFProvider. The class in error will be ignored and WSIF will continue locating other pluggable providers.

WSIF0004E: WSDL contains an operation type “{0}” which is not supported for “{1}”

Explanation: Parameters: {0} name of the operation type specified. {1} name of the portType for the operation. An operation type which is not supported has been specified in the WSDL.

User Response: Remove any operations of the unsupported type from the WSDL. If the operation is required then make sure all messages have been correctly specified for the operation.

WSIF0005E: An error occurred when invoking the method “{1}” . (“{0}”)

Explanation: Parameters: {0} name of communication type. For example EJB or Apache SOAP. {1} name of the method that failed. An error was encountered when invoking a method on the Web service using the communication shown in brackets.

User Response: Check that the method exists on the Web service and that the correct parts have been added to the operation as described in the WSDL. Network problems might be a cause if the method is remote and so check any required connections.

WSIF0006W: Multiple WSIFProvider found supporting the same namespace URI “{0}” . Found (“{1}”)

Explanation: Parameters: {0} the namespace URI. {1} a list of the WSIFProvider found.. There are multiple org.apache.wsif.spi.WSIFProvider classes in the service provider path that support the same namespace URI.

User Response: A following WSIF0007I message will be issued notifying which WSIPFProvider will be used. Which WSIFProvider is chosen is based on settings in the wsif.properties file, or if not defined in the properties, the last WSIFProvider found will be used. See the wsif.properties file for more details on how to define which provider should be used to support a namespace URI.

WSIF0007I: Using WSIFProvider “{0}” for namespaceURI “{1}”

Explanation: Parameters: {0} the classname of the WSIFProvider being used. {1} the namespaceURI the provider will be used to support.. Either a previous WSIF0006W message has been issued or the SetDynamicWSIFProvider method has been used to override the provider used to support a namespaceURI.

User Response: None. See also WSIF0006W.

WSIF0008W: WSIFDefaultCorrelationService removing correlator due to timeout. ID:“{0}”

Explanation: Parameters: {0} the ID of the correlator being removed from the correlation service. A stored correlator is being removed from the correlation service due to its timeout expiring.

User Response: Determine why no response has been received for the asynchronous request within the timeout period. The wsif.asyncrequest.timeout property of the wsif.properties file defines the length of the timeout period.

WSIF0009I: Using correlation service - “{0}”

Explanation: Parameters: {0} the name of the correlation service being used. This identifies the name of the correlation service that will be used to process asynchronous requests.

User Response: None. If a correlation service other than the default WSIF supplied one is required, ensure that it is correctly registered in the JNDI java:comp/wsif/WSIFCorrelationService namespace.

WSIF0010E: Exception thrown while processing asynchronous response - “{0}”

Explanation: Parameters: {0} the error message string of the exception. While processing the response from an executeRequestResponseAsync call an exception was thrown.

User Response: Use the exception error message string to determine the cause of the error. The WSIF trace will have more details on the error including the exception stack trace.

WSIF0011I: Preferred port “{0}” was not available

Explanation: Parameters: {0} the user’s preferred port. The preferred port set by the user on org.apache.wsif.WSIFService is not available

User Response: None unless this message appears for long periods of time in which case the user might want to pick a different port as their preferred port.

WSIF - Known restrictions:

This topic lists the main known restrictions that apply when using WSIF.

Threading

WSIF is not thread-safe.

External Standards

WSIF supports:

- SOAP Version 1.1 (not 1.2 or later).
- WSDL Version 1.1 (not 1.2 or later).

WSIF does not provide WS-I compliance, and it does not support the Java API for XML-based Remote Procedure Calls (JAX-RPC) Version 1.1 (or later).

Full schema parsing

WSIF does not support full schema parsing. For example, WSDL references in complex types in the schema are not handled, and attributes are not handled.

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

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
- Terminology
- Getting started with UDDI Registry
- Setting up and deploying a new UDDI Registry
- Removing and reinstalling a UDDI Registry

- Configuring the UDDI Registry Application
- Managing the UDDI Registry
- UDDI Registry Client Programming
- UDDI Registry Security
- 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) specifications define 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.

The UDDI specifications define 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 specifications. It is a component of WebSphere Application Server Version 6.

A critical component of IBM's on-demand Service Oriented Architecture, 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 had there choices:

1. Would not have known about the similar application
2. Knew about it but could not reuse due to technical barriers
3. Knew about it and reused 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:

- Supports the UDDI Version 3.0 specifications and also supports the Version 2.0 standard API.
- 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 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 department, company, consortium and so on.

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 (or a node within a multi-node 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 will allow 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 programmatically manage UDDI-specific aspects of the WebSphere environment.

Multi-database support

The UDDI data is persisted to a registry database. In WebSphere Application Server Version 6 the databases that are supported are DB2, Cloudscape, and Oracle. Support is provided for remote 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 product.

UDDI Utility Tools

UDDI Utility Tools allows 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/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 the WebSphere Application Server is referred to as **WAS_HOME**. The default locations are:

Windows

WAS_HOME
C:\Program Files\IBM\WebSphere\AppServer\

Linux/Solaris/HP Platforms

WAS_HOME
/opt/IBM/WebSphere/AppServer/

AIX Platform

WAS_HOME
/usr/IBM/WebSphere/AppServer/

z/OS Platform

WAS_HOME
/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 with 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 have to set values for certain properties and to 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 uddiDeploy.jacl, in which case the UDDI database will be a Cloudscape database. The second is to run the supplied SQL script insert_default_database_indicator against the UDDI database, in which case the UDDI database can be Cloudscape, DB2 or Oracle. 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 published). Each UDDI publisher will have a set 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 (WAS) 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
2.1.x	• UDDI Version 1 • UDDI Version 2	5.1.x
3.0.2	• UDDI Version 1 • UDDI Version 2.0.4 (APIs), Version 2.0.3 (data structures) • UDDI Version 3.0.2	6.0

UDDI Registry Security

The IBM UDDI V3 Registry is designed to exploit the advantages of WebSphere security, and it supports the UDDI v1,v2 security features and UDDI V3 Security API.

For production use, it is recommended that the IBM UDDI V3 Registry is configured to use WebSphere security and avoid use of UDDI V1,V2 security features and the V3 Security API.

But, for solutions with a strong preference for UDDI security, the IBM UDDI V3 Registry can be configured to enable its use, both when WAS security is enabled and when it is disabled.

Configuring UDDI to use WAS security

To exploit WebSphere security features WebSphere Application Server (WAS) security must be enabled. When security is enabled, the default settings in UDDI v3 Application and Web deployment descriptors result in the following features:

Authentication

UDDI exploits WAS security Role Mapping by mapping users of UDDI Publish services to the special AllAuthenticatedUsers role to ensure that only valid WebSphere registered users can access these services. This is the default setting for:

- V1,2 SOAP Publish service (SOAP_Publish _User)
- EJB Publish service (EJB_Publish_Role)
- V3 GUI Publish service (GUI_Publish_User)
- V3 Publish service (V3SOAP_Publish_User_Role)
- V3 Custody Transfer service (V3SOAP_CustodyTransfer_User_Role)
- V3 Security service (V3SOAP_Security_User_Role)

The following services are mapped to the special role Everyone and are not protected:

- V1,2 SOAP Inquiry service (SOAP_Inquiry_User)
- EJB Inquiry service (EJB_Inquiry_Role)
- V3 GUI Inquiry service (GUI_Inquiry_User)
- V3 SOAP Inquiry service (V3SOAP_Inquiry_User_Role)

This restriction can be managed using WebSphere Administrative Console features:**Applications > Enterprise Applications> UDDI V3 Registry > Additional Properties > Map security roles to users/groups**.

Further information is also available in: Configuring UDDI Security Roles.

Data Confidentiality

UDDI also exploits the advantages of the WAS security feature enforcing service data confidentiality (’security constraint‘,’user-data-constraint‘, ’transport-guarantee‘ = ’CONFIDENTIAL‘ requiring the use of HTTPS) for the following services:

- V1,2 SOAP Publish service (SOAP_Publish _User)
- EJB Publish service (EJB_Publish_Role)
- V3 GUI Publish service (GUI_Publish_User)
- V3 Publish service (V3SOAP_Publish_User_Role)
- V3 Custody Transfer service (V3SOAP_CustodyTransfer_User_Role)
- V3 Security service (V3SOAP_Security_User_Role)

By default the following services are do not enforce data confidentiality (’security constraint‘, ’user-data-constraint‘, ’transport-guarantee‘ = ’NONE‘ not requiring the use of HTTPS, HTTP is OK):

- V1,2 SOAP Inquiry service (SOAP_Inquiry_User)
- EJB Inquiry service (EJB_Inquiry_Role)
- V3 GUI Inquiry service (GUI_Inquiry_User)
- V3 SOAP Inquiry service (V3SOAP_Inquiry_User_Role)

For information on how to manage the data confidentiality restriction, please refer to ’Configuring SOAP API and GUI services‘ section on user data constraint transport guarantee.

Configuring UDDI to use UDDI security

It is possible to exploit UDDI security features when WebSphere Application Server (WAS) security is enabled or disabled. Depending on the WAS security enabled/disabled setting, different configuration is required and different behavior achieved. While useful for test, it is not anticipated that WAS security is disabled for production configurations.

UDDI security with WAS Security enabled

When WAS security is enabled, to use the UDDI v1//2 (publish) security features (get_AuthToken,discard_AuthToken) or the UDDI v3 security API (get_AuthToken, discard_AuthToken) it is necessary to do the following:

- • set the WAS security Role Mappings to Everyone for:
  - V1,2 SOAP Publish service (SOAP_Publish _User)
  - V3 Publish service (V3SOAP_Publish_User_Role)
  - V3 Custody Transfer service (V3SOAP_CustodyTransfer_User_Role)
  - V3 Security service (V3SOAP_Security_User_Role)

  As identified above this can be managed using the WebSphere Administrative Console: **Applications > Enterprise Applications> UDDI V3 Registry > Additional Properties > Map security roles to users/groups**
- • ensure that UDDI Policy is set to require the use of AuthTokens for the Publish and CustodyTransfer API services (it is implicit for V1/2 Publish).

  This can be managed using the WebSphere Administrative Console: **UDDI>UDDI Nodes>uddinodename>Policy Groups>APIs>‘authorization for publish‘, ’authorization for custody transfer‘**

With this configuration, no Security Role authentication restriction is imposed, but the credentials associated with the authToken are authenticated by WebSphere. Further information is also available in: Configuring UDDI Security Roles.

Note, that when WAS security is enabled, WAS data confidentiality management is independent of UDDI security and is managed as described above.

UDDI security with WAS Security disabled

With WAS Security disabled, neither WAS Security Roles nor data confidentiality constraints apply. This mode may be useful for test UDDI Registry configurations.

In this mode, UDDI V1/2 security features are active, and UDDI V1/2 authTokens should be used on UDDI V1/2 publish requests. Publisher requesting an authToken or using an authToken do have to be a registered WAS users.

With respect to UDDI V3, the use of the UDDI V3 Security API, and the use of authTokens with V3 Publish and Custody Transfer APIs is optional. If required, the following configuration steps are necessary:

- use the WebSphere Administrative Console to identify that use of authInfo is required: **UDDI>UDDI Nodes>uddinodename >General Properties>** check ’Use authInfo credentials if provided‘
- ensure that UDDI Policy is set to require the use of AuthTokens for the Publish and CustodyTransfer using the WebSphere Administrative Console: **UDDI>UDDI Nodes>uddinodename>Policy Groups>APIs>check ‘authorization for publish‘ and ’authorization for custody transfer**

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:

- Default user name -used when WAS Security is disabled and no authToken data is supplied (see WebSphere Administrative Console **> UDDI>UDDI Nodes>uddiNodeName >General Properties> Default user**).
- Authorization for Inquiry, Authorization for Publish, Authorization for Custody Transfer – as described above, checked to require use of authTokens when not 'overridden' by WAS Security Role AllAuthenticatedUsers (see WebSphere Administrative Console**> UDDI>UDDI Nodes>uddiNodeName > Policy Groups>APIs>'authorization for Inquiry', 'authorization for publish' and 'authorization for custody transfer'**).
- Use authInfo credential if provided – as described above, applicable when WAS security is disabled (**UDDI>UDDI Nodes>uddinodename >General Properties> 'Use authInfo credentials if provided'**).
- Authentication token expiry period – to determine the length of idle time allowed before an authToken becomes invalid (see WebSphere Administrative Console > **UDDI>UDDI Nodes>uddiNodeName >General Properties> Authentication token expiry period**).
- Key space requests require digital signature – determines whether all tModel:keyGenerator requests for key space must be signed. To understand key space refer to the Entity Keys Section, to manage this setting (see WebSphere Administrative Console **> UDDI>UDDI Nodes>uddiNodeName > General Properties> Key space requests require digital signature**).

In addition to Security specific policies and properties, awareness of some Keying Policy and User Policies also influences publish behavior:

- Keying Policy – Registry key generation. If this is set, publishers may request key space and if successful publish with publisher assigned keys (see WebSphere Administrative Console **> UDDI>UDDI Nodes>uddiNodeName >Policy Groups> UDDI Keying**).
- Automatically register UDDI publishers – UDDI requires publisher entitlements to be set, before allowing any publish request. This option automatically registers Users with default entitlements (see WebSphere Administrative Console **> UDDI>UDDI Nodes>uddiNodeName >General Properties> Automatically register UDDI Publishers**).
- UDDI Publishers – if the 'Automatically register UDDI publishers' option is not selected, then Users (and their entitlements) can be registered (see WebSphere Administrative Console **> UDDI>UDDI Nodes>Additional properties> UDDI Publishers**).See also the topics on Create UDDI Publishers and UDDI Publisher settings.
- UDDI tier limits – UDDI V3 publish tier limits management (enforcing publisher tier limit) can be activated (see WebSphere Administrative Console **> UDDI>UDDI Nodes> uddiNodeName>General Properties>Use TierLimits**). Also refer to the 'Use tier limits' UDDI property.
- If the above option is checked, then it relevant to configure Tiers (see WebSphere Administrative Console **> UDDI>UDDI Nodes> uddiNodeName>Addition Properties>Tiers**) also to set any registered UDDI Publishers Tier entitlement (see WebSphere Administrative Console**> UDDI>UDDI Nodes> uddiNodeName>Addition Properties>UDDI Publishers**).

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 business identifier feature is not supported.
  - Find technical model (tModel)
    - The business identifier feature is not supported.
  - Add business
    - You must supply the business contact as a name and role (no other information is supported).
  - Add technical model (tModel)
    - You can enter the overview URL, but only with one description in English.
  - 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.

The UDDI user console is split into three distinct areas. At the top of the screen are buttons that activate various functions in the areas below this bar. These buttons are:

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 two tabs mentioned above, the Find and Publish tabs.

Find tab

The Find tab 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 box for entering the name to search for and, beneath this, a ’Find’ link to start the search. Comments are provided to show the user the wildcard character. The results of clicking the ’Find’ link are shown in the detail frame to the right.

Beneath the Quick Find is a section for **Advanced Find** functions which enables the user 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 the user can specify search criteria. To initiate a Find, the user must first enter a search path (the % wildcard may be used) and then click the Add Name link to enter the search. Then click the ’Find Services’ (or ’Find Businesses/Find technical models) link below to initiate the Find operation. The **Locator** section has a link (marked in blue with the words ″**Show category tree**″) which displays the tree from which the user can select categories (or taxonomies). This is shown in the left-hand frame. In the advanced search form there are two links 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 tab is split into three distinct sections.

1. **Quick Publish Function**

   The top part is a **Quick Publish** section to allow the user 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 box for entering the name to assign to the selected entity and, beneath this, a blue **Publish** link to publish the entity. The results of clicking the **Publish** link 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 the user may enter details about the entity they want to publish. Similarly the **Locator** section allows taxonomies to be shown in the left frame from which the user can select categories.

   Following entry of the relevant details on the **Advanced Publish** section, the user must click the **Publish Business** bar in order for the business to be published 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

   Users can **Edit** or **Delete** businesses owned by them by clicking the appropriate links in the **Actions** column.

   After an **Edit** or **Delete** function has been completed, the user *must* click the **Update Business** bar to publish the service to 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 a Service** link in the **Services** column of the **Registered Businesses** section.

   Click the **Add Service** button to publish the service to the UDDI Registry.

   Technical Models

   Technical Models owned by the user 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 [441120]. 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 the Publish Business button at the bottom of the form or at the top. A page is displayed showing the business details and the business is published to the UDDI Registry.

2. **Add a Service**

   From the Publish tab, there is a **Show owned entities** link shows the businesses in the UDDI Registry owned by the current user. 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 completion 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. A tModel can only have one name so there is no blue Add link next to the Technical Model Name field. 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 you can use to programmatically manage UDDI nodes.

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 WAS_HOME/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 policyGroup
     Id) [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 specied 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 bas e 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.

get Properties

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(
                       "C:/valuesets/myvalueset.txt",
                       "uddi:cell:node:server:myValueSet");
   ```

changeValueSetTModelKey

Any value set values that were allocated to one value set tModel are allocated to the new value set tModel.

1. Invoke changeValueSetTModelKey with old and new tModel keys:

```
uddiNode.changeValueSetTModelKey(
            "uddi:cell:node:server:myValueSet",
            "uddi:cell:node:server:myNewValueSet");
```

unloadValueSet

Unloads values for a value set with the given tModel key.

1. Invoke unloadValueSet:

```
uddiNode.unloadValueSet("uddi:myValueSet");
```

isExistingValueSet

Determines if value set data exists for the given tModel key.

1. Invoke isExistingValueSet and display result:

```
boolean exists = uddiNode.isExistingValueSet(
                "uddi:uddi.org:ubr:categorization:naics:2002");
System.out.println("NAICS 2002 is a value set: " + exists);
```

User-defined value set support in the UDDI registry:

In UDDI Version 2 this was called 'Custom Taxonomy Support'.

Data is worthless if it is lost within a mass of other data and cannot be distinguished or discovered. If a client of UDDI cannot effectively find information within a registry, the purpose of UDDI is considerably compromised. Providing the structure and modeling tools to address this problem is at the heart of UDDI's design. The verification of data within UDDI is core to its mission of description, discovery and integration. It achieves this by several means.

It allows users to define multiple value sets that can be used in UDDI. In such a way, multiple classification schemes can be overlaid on a single UDDI entity. This capability allows organizations to extend the set of such systems UDDI registries support. One is not tied to a single system, but can rather employ several different classification systems simultaneously.

While default value sets are shipped with the product, the UDDI Version 3 Private 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.

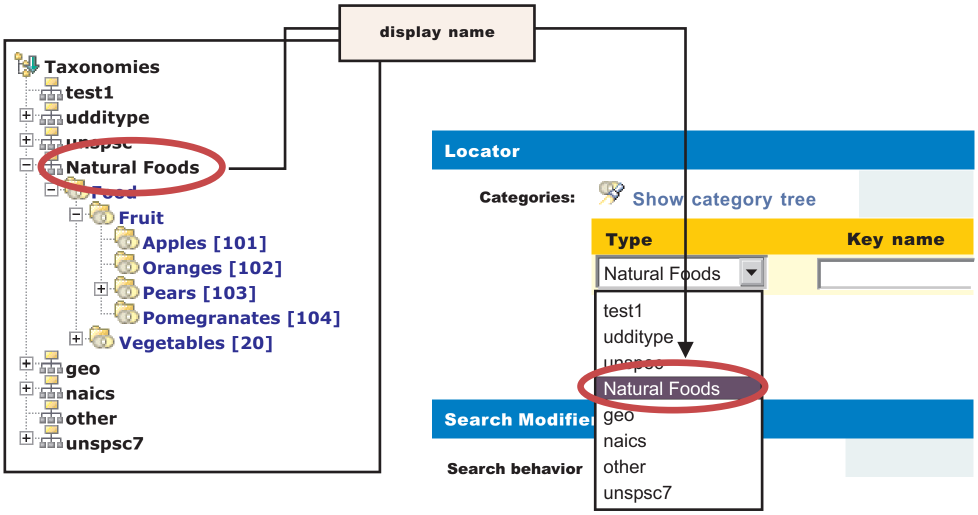
Publish a tModel to the IBM WebSphere UDDI Registry with a categoryBag containing keyedReferences as follows:

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 .bat (for Windows) or UDDIUserDefinedValueSet.sh (for Unix platforms) is located in the WAS_HOME/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.xxx -load valuesetdata.txt
    uddi:a708b8a7-35b5-451c-aafc-718ae071fcfe -columnDelimiter %
```

where .xxx is .bat for Windows or .sh for Unix platforms.

Move value set data from one checked tModel to another on a UDDI Registry in a Network Deployment:

```
UDDIUserDefinedValueSet.xxx –newKey uddi:a708b8a7-35b5-451c-aafc-718ae071fcfe
    uddi:b819c9b8-46c6-562d-bb0d-829bf1820d0f –host depmanagerhost.ibm.com –port
    8879 –node uddinode –server uddiserver -override
```

where .xxx is .bat for Windows or .sh for Unix platforms.

Unload a value set from a tModel from a server with security turned on supplying the connection and security parameters in myproperties.properties file, but supplying the server and password arguments on the command line (which augment or override those contained in the properties file):

```
UDDIUserDefinedValueSet.xxx –unload uddi:b819c9b8-46c6-562d-bb0d-829bf1820d0f
    –server uddiserver –properties myproperties.properties
    –password myrealpassword
```

where .xxx is .bat for Windows or .sh for Unix platforms.

The configuration file, if specified by the optional **-properties** parameter, determines a number of optional parameters. These parameters can be specified on the command line and, if so, override the values in the properties file. These parameters are largely JMX connection parameters and security parameters.

The string.delimiter is typically used where a description value contains the same character as the column delimiter character. For example, if the column.delimiter was set to ',' (a comma), and there was a value set description value of 'Fruits, citrus', you could include this in the value set data file by setting the string.delimiter to ″ (double quote) and enclosing the description in quotes: 'Fruits, citrus'. Note that the quote character is escaped with a backslash ('\') to indicate the literal character is to be used.

If an attempt is made to load a value set to a tModel that has existing value set data, a warning message is given. To override this error provide the **-override** argument. This argument is also required if moving value set data to a new tModel using **-newKey** where the tModel is **checked**, and also unloaded value set data for a **checked** tModel.

Command line arguments and example data	Property and example data	Comments
-columnDelimiter #	column.delimiter=#	Column delimiter used in value set data files
-stringDelimiter \″	string.delimiter=\″	Field delimiter (must be different to the column.delimiter value
-host ibm.com	host=ibm.com	Host name of the system running Deployment Manager or Application Server
-port 8880	port=8880	SOAP port number of Deployment Manager or Application Server
-node ibmNode	node=ibmNode	Name of the Node running the server with the UDDI Registry
-server server1	server=server1	Server running the UDDI Registry
-userName ibmuser	security.username=ibmuser	User name. Required if WebSphere security is turned on

-password mypassword	security.password=mypassword	Password
-trustStore /TrustStoreLocation	security.truststore=/TrustStoreLocation	Truststore file location
-keyStore ibmkeystore	security.keystore=ibmkeystore	Keystore name
-trustStorepassword trustpass	security.truststore.password=trustpass	Truststore password
-keyStorePassword keypass	security.keystore.password=keypass	Keystore password

Set the value set to supported

Use the 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/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 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 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
- Update existing entities in a target UDDI Registry, including child entities
- Delete selected entities from a target UDDI Registry

UDDI Utility Tools can be used by running the UDDIUtilityTools.jar file. This file is located in the WAS_HOME/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

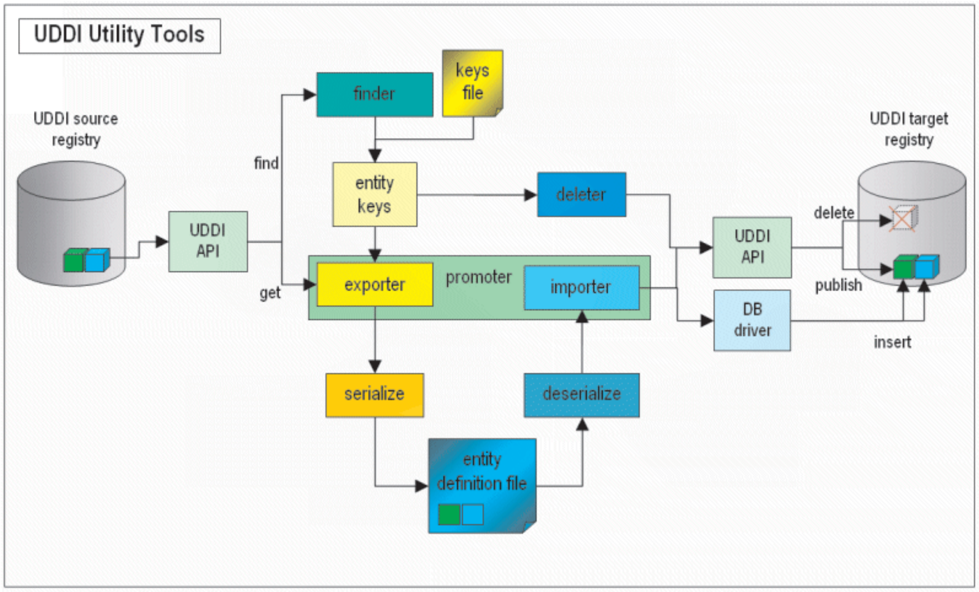
Deletes the specified entities from the target UDDI Registry. The entities to delete are specified as an entity type, or a list of entity types, and keys, in the same way as for the export function.

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 WAS_HOME/UDDIReg/scripts directory.

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     #
###############################################
classpath=.;c:/Progra~1/WebSphere/utilitytools/UDDIUtilityTools.jar;
C:/WebSphere/AppServer/lib/soap.jar;C:/WebSphere/AppServer/java/jre/lib/ext/mail.jar;
C:/WebSphere/AppServer/java/jre/lib/ext/ibmjsse.jar;
C:/WebSphere/AppServer/java/jre/lib/ext/activation.jar;
C:/promoter/uddi4jv2.jar;C:/WebSphere/AppServer/lib/xerces.jar;
C:/WebSphere/AppServer/lib/j2ee.jar;"C:/Program Files/SQLLIB/java12/db2java.zip"

 (ALL the above is on ONE line - shown here as broken for clarity)

###############################################
# SOAP entry points for source UDDI           #
###############################################
fromInquiryURL=http://localhost:9080/uddisoap/inquiryapi
fromGetURL=http://localhost:9080/uddisoap/get

###############################################
```

```
 # SOAP entry points for target UDDI          #
##############################################
toInquiryURL=http://localhost:9080/uddisoap/inquiryapi
toPublishURL=http://localhost:9080/uddisoap/publishapi

##############################################
# UDDI Registry user information             #
#                                            #
# Note: this must match the user information #
# that was used to publish the entities on   #
# the target UDDI registry.                  #
##############################################
userID=UNAUTHENTICATED
password=NONE

##############################################
# Configuration for destination UDDI DB      #
##############################################
dbDriver=COM.ibm.db2.jdbc.app.DB2Driver
dbUrl=jdbc:db2:uddi30
dbUser=db2admin                    (Your dbUser id)
dbPasswd=db2admin                  (Your dbPassword)

##############################################
# 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)
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

UDDI Utility Tools must have the following jar files available. Their locations should be specified in the classpath property in the UDDI Utility Tools properties file:

UDDIUtilityTools.jar
This is the tools jar itself. It MUST appear on the classpath.

uddi4jv2.jar
This is the UDDI4J classes and can be found in <WAS_HOME>/lib.

j2ee.jar
This contains some required J2EE classes.

soap.jar
This is the Apache SOAP implementation.

xerces.jar
This is the xerces XML parser.

DbDriver
This is the default driver needed to allow UUT to connect to your target database. For example, for DB2, it would be `$DB2_HOME/DB2java.zip`, for Oracle, it would be `$ORACLE_HOME/jdbc/lib/ojdbc14.jar` and for Cloudscape you need two jars. These are `WAS_HOME/cloudscape/lib/otherJars/db2jcc.jar` and `WAS_HOME/cloudscape/lib/db2jcc_license_c.jar`. The respective dbURL's and Drivers are: for DB2 jdbc:db2:*databasename* and COM.ibm.db2.jdbc.app.DB2Driver, for Oracle thin client jdbc:oracle:thin:@*host:*1521:*databasename* (the number specified is the default port number and this may change depending on your Oracle setup) and Oracle.jdbc.driver.OracleDriver, for Cloudscape jdbc:db2j:net://host:1527/*databasename* (databasename includes the path) and com.ibm.db2.jcc.DB2Driver. Note that if your destination database is Cloudscape you will need to make it network enabled so that it can handle multiple connections. See Configuring Cloudscape Version 5.1.38 for details on how to do this. For information on how to set up Cloudscape for multiple connections, see:

```
http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp?topic=
   /com.ibm.websphere.base.doc/info/ae/ae/ tdat_cloudscape_setup.html
```

Apache SOAP, and therefore UDDI Utility Tools, has a requirement of having activation.jar and mail.jar java extensions available. These should NOT be placed on the classpath but rather the ext folder of the JRE that is used to run the tool. If SSL is needed then ibmjsse.jar must also be in the ext folder. If you are using the JRE as supplied with IBM WebSphere Application Server then these extensions will already be in place and no further action is necessary.

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 the correct level of java is appropriate by setting the PATH statement to the level of java supplied with WebSphere. For example, from the command line, type:

On Windows:

```
set PATH=WAS_HOME\java\bin;%PATH%
```

On Unix and Linux platforms:

```
export PATH=WAS_HOME/java/bin:$PATH
```

If you are using DB2 on Unix and Linux platforms run the db2profile script before issuing the java command to start UDDI Utility Tools. This script is located within the DB2 instance home directory under sqllib and is invoked by typing:

```
. /$DB2_HOME/db2profile
```

Note: In the above example, notice that the '.' is followed by a single space character.

Note: On Unix and Linux platforms the DB2 user **must** have a db2profile at $HOME/sqllib/db2profile.

UDDI Utility Tools can be started using:

java - jar UDDIUtilityTools.jar <function> [options]
using a specified properties file that sets up classpath and other parameters, or it can be called using:

java CommandLineProcessor
where CommandLineProcessor is the class which processes command line arguments for UDDI Utility Tools, sets up configuration and invokes the appropriate function.

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 C:/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 c:/myKeyFiles/keyFile01.txt
```

As above but also specifying verbose output to appear on the command line.

```
java -jar UDDIUtilityTools.jar -export -keysFile c:/myKeyFiles/keyFile02.txt -v
```

To import the contents of the default EDF specified in a UDDIUtilitiyTools.properties file in the current directory.

```
java -jar UDDIUtilityTools.jar -import
```

As above but also specifying that referenced tModels should be imported into the target registry.

```
java -jar UDDIUtilityTools.jar -import -importReferenced
```

To import the entities from an EDF at the specified location. Note the use of forward slashes even though this is an example on a Windows file system.

```
java -jar UDDIUtilityTools.jar -import -definitionFile c:/myEDFs/entities01.xml
```

To import the entities from the default EDF including referenced tModels. Overwrite specifies that any entities excluding referenced tModels that are found in the target registry should be overwritten.

```
java -jar UDDIUtilityTools.jar -import -overwrite -importReferenced
```

To promote a single service from a source to a target registry using the properties file at a specified location.

```
java -jar UDDIUtilityTools.jar -promote -service 67961D67-330F-4F14-8210-E74A58E710F3
-properties c:/UUT/myUUTProps.properties
```

To promote a set of entities specified in a keys file.

```
java -jar UDDIUtilityTools.jar -promote -keysFile c:/myKeyFiles/keyFile03.txt
```

As above but specifying that existing entities in the target registry get overwritten.

```
java -jar UDDIUtilityTools.jar -promote -keysFile c:/myKeyFiles/keyFile04.txt -overwrite
```

To promote a set of entities specified in a keys file including referenced tModels.

```
java -jar UDDIUtilityTools.jar -promote -keysFile c:/myKeyFiles/keyFile05.txt -importReferenced
```

To promote a set of entities specified in a keys file but also create an EDF containing the promoted entities.

```
java -jar UDDIUtilityTools.jar -promote -keysFile c:/myKeyFiles/keyFile06.txt
-definitionFile c:/myEDFs/entities02.xml
```

To logically delete a single tModel. Note that it is not possible to physically delete tModels.

```
java -jar UDDIUtilityTools.jar -delete -tModel UUID:1E2B9D1E-E53D-4D36-9D46-6CCC176C466A
```

To delete all the entities specified in the keys file. Note that with the exception tModels all other entities will be physically deleted from the target registry.

```
java -jar UDDIUtilityTools.jar -delete -keysFile c:/myKeyFiles/keyFile04.txt
```

A keys file example

Below is an example of the keys that are to be exported, promoted or deleted from the target registry:

```
#
# Keys of entities to be exported, promoted from source registry or deleted from
    target registry
#
# Note: keys must be comma separated and on SAME line
# Note: property names are case sensitive. ('tmodels=' will be ignored)

businesses=97C77097-AC6C-4CA0-A6C4-452F7045C470, 4975E949-581F-4FCA-AD5F-E08280E05F9F
services=BB3864BB-1578-4833-8179-14391F14791F
bindings=
tModels=273F1727-7BFF-4FB5-A1FD-BA5C45BAFD9C
```

Note: If the importReferenced property is set to true, the list of tModels in the referencedTModels section is imported to the target registry. Minimal entities are created if the referencedTModel is new. If the referencedTModel already exists it is never overwritten, regardless of the overwrite property value. This is so that commonly referenced tModels such as categorization tModels do not keep being updated unnecessarily.

Should you need to update a referencedTModel, you must manually move the referencedTModel definition to the tModels section in the entity definition file and set overwrite to true.

Content of the log files

Below shows examples of contents of two of the log files produced by running the tool. Note that some comments have been added in square brackets and in italic to highlight important points within the log file. The first is the messages.log which shows successful and unsuccessful operations for export, import and delete functions:

```
[29/07/04 17:39:57:531 BST] CWUDU0002I: ***** Starting UDDI Utility Tools ****
****** [timestamp and eyecatcher indicate when tool is run]
[29/07/04 17:39:57:531 BST] CWUDU0009I: Exporting entities...
[29/07/04 17:39:57:531 BST] CWUDU0015I: Exported 14 entities.
[29/07/04 17:39:57:531 BST] CWUDU0029I: Serializing...
[29/07/04 17:39:57:531 BST] CWUDU0030I: Serialized entities.
[29/07/04 17:39:57:531 BST] CWUDU0016I: Importing entities...
[29/07/04 17:39:57:531 BST] CWUDU0124I: Created tModel minimal entity with tModelKey
   [uuid:667e2766-4781-4151-b3a0-809f7180a096].
[29/07/04 17:39:57:531 BST] CWUDU0121I: Created business minimal entity with
   businessKey [263f5526-8708-4834-9f5d-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=c:\temp/MigToolFiles/Results/Promote_api_EDF_1.xml

ExportConfiguration:
  fromGetURL=http://yottskry:9080/uddisoap/
  fromInquiryURL=http://yottskry:9080/uddisoap/inquiryAPI

ImportConfiguration:
  overwrite=true
  uddiEntityDefinitionFile=c:\temp/MigToolFiles/Results/Promote_api_EDF_1.xml
  importReferencedEntities=true

PublishConfiguration:
  toInquiryURL=http://davep:9080/uddisoap/inquiryAPI
  toPublishURL=http://yottskry:9080/uddisoap/publishAPI
  userID=Publisher1
  trustStoreFileName=c:/WebSphere600/AppServer//etc/DummyClientTrustFile.jks
  secureConnection=false

DatabaseConfiguration:
  dbDriver=COM.ibm.db2.jdbc.app.DB2Driver
  dbURL=jdbc:db2:UDDI30
  dbUser=db2admin

LoggerConfiguration:
  messageStream=null
  messageLogFileName=c:\temp/MigToolFiles/logs/message.log
  traceLogFileName=c:\temp/MigToolFiles/logs/trace.log
  traceLevel=3
  verbose=true


[29/07/04 17:39:57:531 BST] < com.ibm.uddi.promoter.PromoterAPI()
[29/07/04 17:39:57:531 BST] ********** Starting UDDI Utility Tools **********
[29/07/04 17:39:57:531 BST] > com.ibm.uddi.promoter.PromoterAPI.setUddiEntities()
[29/07/04 17:39:57:531 BST] > com.ibm.uddi.promoter.export.KeyFileReader()
[29/07/04 17:39:57:531 BST]   com.ibm.uddi.promoter.export.KeyFileReader() loaded
    tModel keys [ log entries without a '>' or '<' are status messages only ]
[29/07/04 17:39:57:531 BST]   com.ibm.uddi.promoter.export.KeyFileReader() loaded
    business keys
[29/07/04 17:39:57:531 BST]   com.ibm.uddi.promoter.export.KeyFileReader() loaded
    service keys
[29/07/04 17:39:57:531 BST]   com.ibm.uddi.promoter.export.KeyFileReader() loaded
    binding keys
[29/07/04 17:39:57:531 BST] > com.ibm.uddi.promoter.UddiEntityKeys()
[29/07/04 17:39:57:531 BST] < com.ibm.uddi.promoter.UddiEntityKeys() [the '<'
    indicates exit from the constructor]
[29/07/04 17:39:57:531 BST]   com.ibm.uddi.promoter.export.KeyFileReader() removed
    duplicate, empty and null keys
[29/07/04 17:39:57:531 BST] < com.ibm.uddi.promoter.export.KeyFileReader()
```

```
[29/07/04 17:39:57:531 BST] < com.ibm.uddi.promoter.PromoterAPI.setUddiEntities()
[29/07/04 17:39:57:531 BST] > com.ibm.uddi.promoter.PromoterAPI.deleteEntities()
[29/07/04 17:39:57:531 BST] > com.ibm.uddi.promoter.publish.EntityDeleter()
[29/07/04 17:39:57:531 BST] < com.ibm.uddi.promoter.publish.EntityDeleter()
[29/07/04 17:39:57:531 BST] > com.ibm.uddi.promoter.UDDIClient()
[29/07/04 17:39:57:531 BST]   com.ibm.uddi.promoter.UDDIClient() client type: 1
```

Starting UDDI Utility Tools through the API

UDDI Utility Tools provides a public API to functions for exporting, importing, promoting, finding and deleting UDDI entities. All of these functions can be invoked by using the PromoterAPI class. Usage of this class to perform these functions is typically to:

1. Create a Configuration object and populate it from a Properties object or from a configuration properties file.
2. Create a PromoterAPI object passing the Configuration in the constructor.
3. For keys based functions (export, delete and promote), set the keys by supplying a UDDIEntityKeys object, the location of the keys file, or, for one entity, by specifying an entity type and a key value.
4. Invoke the corresponding method for the function required: exportEntities, promoteEntities(boolean), importEntities, deleteEntities or extractKeysFromInquiry(FindTModel, FindBusiness, FindService, FindBinding, FindRelatedBusinesses).

There is some sample code for UDDI Utility Tools, demonstrating usage of the API classes, available from Samples Central.

The ″low-level″ API classes and methods have been deprecated in this release. Refer to the Javadoc welcome page 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″ in the information center.

Cloudscape Restriction

The ’export’ function when referencing a source registry with a Cloudscape database is supported. However, the ’import’, ’promote’ and ’delete’ functions are not supported when referencing a target registry because of a limitation with the UDDI Registry when working with a Cloudscape database.

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, when security is enabled, 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 &#8209;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 WAS_HOME/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 WAS_HOME/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.registry.xml.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. This 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 the 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 name>#<parent element value>
   ```

Token	Description
<taxonomy ID>	The taxonomy ID is the same for every element of a taxonomy.
<element value>	The taxonomy ID is the same for every element of a taxonomy.
<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 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 Concept tree

Because there is only one 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
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.

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.

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.

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: <true|false>

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 itdoes 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:

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, txnInit, failed getting persister control

Explanation: NodeManager initialization failure.

User Response: Contact the IBM Customer Service Center.

CWUDN0005E: Error, NodeManager, txnIInit, UDDIFatalErrorException during Init

Explanation: NodeManager initialization failure.

User Response: Contact the IBM Customer Service Center.

CWUDN0006E: Error, NodeManager, txnIInit, UDDIException during initialization.

Explanation: NodeManager initialization failure.

User Response: Contact the IBM Customer Service Center.

CWUDN0007E: Error, NodeManager, txnIInit, 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, txnlInit, Throwable releasing connection

Explanation: NodeManager initialization failure.

User Response: Contact the IBM Customer Service Center.

CWUDN0011E: Error, NodeManager, txnlDestroy, failed getting Persister control

Explanation: NodeManager application stop or removal error.

User Response: Contact the IBM Customer Service Center.

CWUDN0012E: Error, NodeManager, txnlDestroy, UDDI Exception during destroy

Explanation: NodeManager application stop or removal error.

User Response: Contact the IBM Customer Service Center.

CWUDN0013E: Error, NodeManager, txnlDestroy, rollback Exception

Explanation: NodeManager application stop or removal error.

User Response: Contact the IBM Customer Service Center.

CWUDN0014E: Error, NodeManager, txnlDestroy, 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.

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.

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 C:/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 sasme 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.

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 enumberation, 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 (that is, it is not a taxonomy Concept), that is invalid. 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 bythe 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.

Removing and reinstalling UDDI Registry

An IBM WebSphere UDDI Registry node consists of a WebSphere application, a store of data (using a relational database management system - 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 exits irrespective of the UDDI application.

With these facts in mind, consider the following options:

- • To remove a node from a WebSphere Application Server the database need not be deleted. Only the UDDI application and any associated resources such as the data source used (and J2C Authentication Data if used) need to be removed as the data in the UDDI database is separate from the UDDI application.
- • It is not necessary to remove the UDDI application to start a new UDDI Registry node. Instead, a new, replacement node can be created by changing the datasource which the UDDI application uses to access new UDDI database.

Remove the UDDI application if you no longer want a UDDI facility on a particular WebSphere Application Server (the UDDI Registry node may subsequently be move 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.

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, or to remove a UDDI application or to remove a UDDI Registry node, see ″Removing a UDDI Node″ in the information center.

Removing a UDDI Registry node

- Remove 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.
- Delete 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 remove 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, which was installed into the WAS_HOME\bin directory when you installed WebSphere

   At a command prompt enter:

   ```
   wsadmin -f uddiRemove.jacl
                  nodename
                  servername
                  [default]
                  > removeuddi.log
   ```

   where
   - *nodename* and *servername* are the name of the WebSphere node and application server in which the UDDI application is deployed (these are the names that you specified when you ran uddiDeploy.jacl to install the UDDI application).
   - [default] is optional. Specifying *default* will remove the UDDI Cloudscape datasource but **not** the UDDI Cloudscape database. This is only applicable if *default* was used when the uddiDeploy.jacl script was run to deploy the UDDI Registry, and is only applicable in a standalone application server environment.
   - by default output will go to the screen, but, optionally, you can specify '> *removeuddi.log*' to direct the output to a log file (where removeuddi.log can be any log name of your choice).

   For example:

   ```
   wsadmin -f uddiRemove.jacl MyNode server1
   ```

   will remove the UDDI application from server server1 running in node MyNode, and will send any messages to the screen.

   Note: If running in a Network Deployment configuration, the default option cannot be used, and 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.
   - For DB2, use the database facilities to delete the UDDI database.
   - For Oracle, delete the schemas IBMUDI30 and IBMUDS30.
   - For Cloudscape, delete the directory tree containing the UDDI database. By default, this will be located in WAS_HOME:/profiles/profile_name/databases/com.ibm.uddi/UDDI30.
3. **Completely Remove 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 uddiRemove.jacl as described above, to remove the UDDI Registry application.

b. Delete the UDDI Registry database, as described above.

c. This last step is only needed if you ran uddiRemove without the default option (use of the default option will perform this step for you, but is only valid if the UDDI Registry was previously deployed using uddiDeploy.jacl with the default option).

   - Delete the UDDI datasource that references the UDDI Registry database (this will have been created when the UDDI Registry was set up), any UDDI JDBC provider that was created (if you did not reuse an existing JDBC provider), and 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.

If you wish to reinstall the UDDI application, follow the instructions below:

1. 1. Execute the following command

   ```
   wsadmin [-conntype none] –f uddiDeploy.jacl <node> <server>
   ```

   which will uninstall the UDDI application and install the uddi.ear located in WAS_HOME/installableApps, resulting in a reinstalled UDDI application.

   Note: No JDBC Providers, Data Sources or J2C Application data will be created, changed or deleted during this process.

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 security roles
- Multiple language encoding support in UDDI
- Customizing the UDDI User Console (GUI)
- Configuring SOAP API and GUI services
- Configuring WebSphere to use HTTPS and SSL

Configuring UDDI Security Roles

Each interface to the UDDI Registry (either through Version 1 and 2 SOAP, Version 3 SOAP, EJB or the GUI) is supplied with two roles:

Publish role(s) and custody transfer
mapped to AllAuthenticatedUsers. By default, this is configured to use SSL (that is HTTPS), but this only applies when WebSphere security is enabled.

Inquiry role
mapped to Everyone. By default, this is configured to use HTTP (that is not SSL).

In addition UDDI Version 3 SOAP has an additional role for version 3 SOAP Custody Transfer, which is mapped in the same way as the Publish Role.

The security role mappings can be altered by users through the Administration Console.

Authentication uses the standard WebSphere facilities and there is no separate registration function for the Registry. If WebSphere security is enabled, you will need to supply your WebSphere userid and password for Publish functions (unless you have changed the supplied Publish role).

You will need to set up WebSphere security configuration to be used by UDDI. It is expected that, for development use, security will be disabled and security will be enabled for production environments.

The V3 SOAP services also support the use of the V3 Security API set get_authToken and discard_authToken, but its use is optional, and defined by a combination of UDDI Policy and Security Roles.

- If security is enabled, and the service's Role is set to AllAuthenticated Users, WebSphere security takes priority over UDDI authentication, but if the Publish (or Custody Transfer) Role is mapped to Everyone and policy is set to require Authorization for publish (or Custody Transfer) (see WebSphere Administration console (UDDI => UDDI Nodes => UDDI NodeID => Policy Groups => APIs => General Properties)) then an authToken is required, and the user and password supplied in get_authToken will be checked by WebSphere.
- If security is disabled, Role Mapping does not apply, and the use of the V3 Security API set is defined by policy. If the 'Use authInfo if provided' option is checked (see WebSphere Administrative console (UDDI => UDDI Nodes => General Properties)) the userid supplied in get_authToken is used (but the password will not be checked). If 'Use authInfo if provided' is not checked the default user, set to UNAUTHENTICATED by default (but configurable, see WebSphere Administrative console (UDDI => UDDI Nodes => General Properties => Default user ID)) is used.

The V1/2 SOAP interface also supports the UDDI API for get_authToken and discard_authToken API but use of this is optional.

- If security is disabled and get_authToken is not called, the default user, UNAUTHENTICATED, is used.
- If security is disabled and get_authToken is called, the specified userid is used (but the password is not checked).
- If WebSphere security is enabled, it takes priority over UDDI authentication, but if the Publish role is mapped to Everyone, get_authToken must be used and the userid and password will be checked by WebSphere.

The Security Roles provided with the UDDI Registry are as follows:

- GUI_Publish_User
- GUI_Inquiry_User
- SOAP_Publish_User
- SOAP_Inquiry_user
- EJB_Inquiry_Role
- EJB_Publish_Role
- V3 SOAP_Inquiry_User_Role
- V3 SOAP_Publish_User_Role
- V3 SOAP_CustodyTransfer_User_Role
- V3 SOAP_Security_User_Role

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 User Console (GUI)

The look and feel of the UDDI console is determined by the styles defined in the *.css* files which are located in the /v3gui.war/theme directory of the installed UDDI Registry application directory. The UDDI Registry application directory will be one of the following, depending on where you have installed the UDDI Registry:

- If you have installed the UDDI Registry into an application server within a deployment manager cell, the directory is WAS_HOME/profiles/installedApps/<currentcell>/UDDIRegistry.<nodename>.<servername>.ear/v3gui.war
- If you have installed the UDDI Registry into a single application server which is not part of a deployment manager cell, the directory is <WAS_HOME>\profiles\<profile name>\installedApps\<node_name>\UDDIRegistry.<nodename>.<servername>.ear\v3gui.war under the installedApps directory of the WebSphere Application Server in which you have installed the UDDI registry application as shown in the example below.

Style class definitions in these files can be edited to alter the overall theme of the user console, including font attributes, layout and colors.

Configuring SOAP API and GUI service

Configuring V1/V2 SOAP API services

You can configure the following Version 1 and Version 2 SOAP interface properties:

- *defaultPoolSize* - 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:

- Edit the active deployment descriptor (web.xml) for the V1/2 SOAP module (soap.war) . This file is located in the soap.war\WEB-INF subdirectory of the saved configuration data for the deployed uddi.ear application. For example:

```
WAS_HOME\AppServer\profiles\<hostname>Profile01\config\cells\
  <hostname>Node01Cell\applications\UDDI V3 Registry\soap.war\WEB-INF\web.xml)
```

- To modify defaultPoolSize for V1/2 Publish modify the ’param-value‘ element in the servlet with ’servlet-name‘ = uddipublish
- To modify defaultPoolSize for V1/2 Inquiry modify the ’param-value‘ element in the ’servlet‘ with ’servlet-name‘ = uddi
- To modify the user data constraint transport guarantee for V1/2 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
- Stop and restart the application server for the changes to take effect

Configuring V3 SOAP API services

You can configure the following Version 3 SOAP interface properties:

- 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, CustodyTransfer and Security APIs is to use HTTPS and Inquiry to use HTTP

To configure these properties after the UDDI application has been installed:

- • Edit the active deployment descriptor (web.xml) for the v3 SOAP module (v3soap.war) . This file is located in the v3soap.war\WEB-INF subdirectory of the saved configuration data for the deployed uddi.ear application. For example:

```
WAS_HOME\AppServer\profiles\<hostname>Profile01\config\cells\
  <hostname>Node01Cell\applications\UDDI V3 Registry\v3soap.war\WEB-INF\web.xml)
```

- To modify the user data constraint transport guarantee for V3 publish which determines whether the V3 Publish service is to be confidential (accessed using HTTPS) or insecure (using HTTP) find the ’security-constraint‘ with ’display-name‘ = ’AxisServlet Publish Resource Collection‘ and set its ’user-data-constraint‘ ’transport-guarantee‘ to CONFIDENTIAL or NONE
- To modify the user data constraint transport guarantee for V3 custody transfer which determines whether the V3 Custody Transfer service is to be confidential (accessed using HTTPS) or insecure (via HTTP) find the ’security-constraint‘ with ’display-name‘ = ’AxisServlet CustodyTransfer Resource Collection‘ and set its ’user-data-constraint‘ ’transport-guarantee‘ to CONFIDENTIAL or NONE
- • To modify the user data constraint transport guarantee for V3 security which determines whether the V3 Security service is to be confidential (accessed using HTTPS) or insecure (via HTTP) find the ’security-constraint‘ with ’display-name‘ = ’AxisServlet Security Resource Collection‘ and set its ’user-data-constraint‘ ’transport-guarantee‘ to CONFIDENTIAL or NONE
- To modify the user data constraint transport guarantee for V3 inquiry which determines whether the V3 Inquiry service is to be confidential (accessed using HTTPS) or insecure (via HTTP) find the ’security-constraint‘ with ’display-name‘ = ’AxisServlet Inquiry Resource Collection‘ and set its ’user-data-constraint‘ ’transport-guarantee‘ to CONFIDENTIAL or NONE
- Stop and restart the application server for the changes to take effect.

Configuring V3 GUI services

You can configure the following Version 3 GUI properties:

- 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:

- Edit the active deployment descriptor (web.xml) for the V3 GUI module (v3gui.war) . This file is located in the v3gui.war\WEB-INF subdirectory of the saved configuration data for the deployed uddi.ear application. For example:

```
WAS_HOME\AppServer\profiles\<hostname>Profile01\config\cells\
    <hostname>Node01Cell\applications\UDDI V3 Registry\v3gui.war\WEB-INF\web.xml)
```

- To modify the user data constraint transport guarantee for V3 publish which determines whether the V3 Publish service is to be confidential (accessed using HTTPS) or insecure (via HTTP) find the ’security-constraint id‘ = ’UDDIPublishSecurityContraint‘ and set its ’user-data-constraint‘ ’transport-guarantee‘ to CONFIDENTIAL or NONE
- To modify the user data constraint transport guarantee for V3 inquiry which determines whether the V3 Inquiry service is to be confidential (accessed using HTTPS) or insecure (via HTTP) find the ’security-constraint id‘ =‘ ’UDDIInquireSecurityConstraint‘ and set its ’user-data-constraint‘ ’transport-guarantee‘ to CONFIDENTIAL or NONE
- Stop and restart the application server for the changes to take effect

Configuring WebSphere to use HTTPS and SSL

To support the use of secure access with the IBM WebSphere UDDI Registry, you must configure WebSphere to use HTTPS and SSL. Refer to Secure Sockets Layer settings for custom properties elsewhere in this Information Center for configuring SSL in WebSphere Application Server. It is assumed throughout the information for the UDDI Registry that, where SSL is used, it has been configured on port 9443.

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 version of WebSphere Application Server and UDDI 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 programmatically monitor and configure UDDI registries, and is explained in ″Using administrative programs (JMX)″ in the information center. 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

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 Name 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.

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.

Important: To be able to manage a UDDI node, the associated UDDI application must be running. If the application is not running you will not see the UDDI node in the list of available choices.

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″ in the information center) 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	A valid domain key for this UDDI node

UDDI node description:

The user supplied description of this UDDI node.

Required	Yes
Data type	String

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:default: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

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.

Note: 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.

Note: 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.

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.

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	Integer
Default	True

Authentication token expiry period:

The period after which authentication tokens are invalidated (in minutes), and a new authToken is required.

Note: 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 - 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.

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.

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. 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 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.

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:

Indicates whether references to this value set are supported by policy in this UDDI node.

Value set settings:

Use this page to view the attributes of a value set in a UDDI node.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id* >**Value Sets** > *value_set_name*.

This page shows the values of keyedReferences in the tModel that represents this value set. 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 topic contains a list of the available tiers for the UDDI node. You can also 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
Range	1 to 255

Description:

The user supplied description of the tier.

Data type	String
Default	Empty
Range	0 to 255

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.

For each of the maximum fields described above, the data is:

Required	Yes
Data type	Integer
Default	0
Range	0 to 2147483647 (for all intents and purposes, unlimited)

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.

Create UDDI Publishers:

Use this page to register existing WebSphere Application Server users as UDDI publishers. Create each UDDI publisher individually using the New button on the UDDI publisher collection page.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id* >**UDDI Publishers** → **New**.

Users known to the application server can be registered as UDDI publishers.

Assign entitlements and a tier to the selected group

UDDI publishers are given permission to perform specific actions with entitlements. Set the entitlements for the selected UDDI publishers with the check-box next to each entitlement.

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 to be created. This should be a user known to the application server.

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 all the entitlement settings are disregarded, irrespective of how they are set.

Allowed to publish keyGenerator with a domain key:

The UDDI publisher has permission to publish tModel:keyGenerator with a domain key.

Allowed to publish keyGenerator with a 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).

Allowed to publish keyGenerator with UUID keys:

The UDDI publisher user has permission to publish tModel:keyGenerator providing a UUID key.

Allowed to publish with UUID key:

The UDDI publisher has permission to publish elements providing a UUID key.

Allowed to subscribe:

The UDDI publisher can register requests to receive notifications of specific registry content changes.

Tier:

The tier to which the selected UDDI publishers are assigned.

UDDI Publisher settings:

Use this page to view and edit the properties of UDDI publishers.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id* >**UDDI Publishers** > *user_name*.

This page shows the entitlements and publication limits tier for a particular UDDI publisher.

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 a domain key', 'Allowed to publish keyGenerator with a derived key', 'Allowed to publish keyGenerator with UUID keys', 'Allowed to publish with UUID keys' and 'Allowed to subscribe') will be disregarded irrespective of how they are set.

Allowed to publish keyGenerator with a domain key:

The UDDI publisher has permission to publish tModel:keyGenerator with a domain key.

Allowed to publish keyGenerator with a 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).

Allowed to publish keyGenerator with UUID keys:

The UDDI publisher user has permission to publish tModel:keyGenerator providing a UUID key.

Allowed to publish with UUID key:

The UDDI publisher has permission to publish elements providing a UUID key.

Allowed to subscribe:

The UDDI publisher can register requests to receive notifications of specific registry content changes.

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 user policy settings”
- “UDDI node API policy settings” on page 1124
- “UDDI data custody policy settings” on page 1124
- “UDDI value set policy” on page 1125
- “UDDI node miscellaneous” on page 1125

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

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

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.

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**.

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.

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.

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.

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.

Note: 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.

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.

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.

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 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)″ in the information center. 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 back-up a Cloudscape based registry, 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.

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

  For restoration we recommend 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, for Oracle see Creating an Oracle database or, for Cloudscape, see Creating a Cloudscape database for details) as per the original creation **except** that the final step (xxxxxx_**70**xxxx.*) must not be run *this will result in a virgin, and almost empty, database into which your backed-up data may be imported.*
  3. Delete all rows in the table IBMUDS30.UDDIDBSCHEMAVER *to avoid a clash between a row inserted by the scripts and the row backed-up using the command:*

     ```
     delete from IBMUDS30.UDDIDBSCHEMAVER
     ```
  4. Restore the previously backed-up data in schemas IBMUDI30 and IBMUDS30

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. See ″Hardware and software requirements″ in the information center for more information.

1. Develop data access applications. Develop your application to access data using the various ways available through the WebSphere Application Server. You can access data through APIs, container-managed persistence beans, bean-managed persistence beans, session beans, or Web components.
2. Assemble data access applications using the assembly tool. Assemble your application by creating and mapping resource references.
3. Prepare for deployment: Ensure that the appropriate database objects are available. Create or configure any databases or tables required, set necessary configuration parameters to handle expected load, and configure any necessary JDBC providers and data source objects for servlets, enterprise beans, and client applications to use.
4. Install the application on your application server.

Resource adapter

A resource adapter is a system-level software driver that a Java application uses to connect to an enterprise information system (EIS).

A resource adapter plugs into an application server and provides connectivity between the EIS, the application server, and the enterprise application.

An application server vendor extends its system once to support the J2EE Connector Architecture (JCA) and is then assured of seamless connectivity to multiple EISs. Likewise, an EIS vendor provides one standard resource adapter with the capability to plug into any application server that supports the connector architecture.

WebSphere Application Server provides the WebSphere Relational Resource Adapter (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.

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 1127
- For information about resource adapters and data access, see “Data access portability features”
- For RRA settings, see “WebSphere relational resource adapter settings” on page 1128
- For information about CMP connection factories, see “Connection factory” on page 1133
- 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 WebSphere 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 use JDBC providers to access data from databases.

For applications that need access to relational databases, the JDBC provider and data source together are functionally equivalent to the J2EE Connector Architecture (JCA) connection factory. The WebSphere Application Server prerequisite Web site has a current list of supported providers. See ″Hardware and software requirements″ in the information center for more information.

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. 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 applications and beyond. 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:

When accessing any database, establishing connections is an expensive operation. *Connection pooling* enables administrators to establish a pool of database connections that applications can share on an application server. When connection pooling capabilities are used, performance improvements up to 20 times the normal results are realized.

Each time a resource attempts to access a backend store (such as a database), the resource must connect to that data store. A connection requires resources to create, maintain, and then release the connection when it is no longer required.

The total data store overhead for an application is particularly high for Web-based applications because Web users connect and disconnect more frequently. In addition, user interactions are typically shorter. Often, more effort is spent connecting and disconnecting than is spent during the interactions. Also, because Internet requests can arrive from virtually anywhere, you can find usage volumes large and difficult to predict.

To help lessen these overhead problems, the 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 resources for future requests.

WebSphere Application Server supports JDBC 3.0 APIs to provide support 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.

Each entity bean transaction requires an additional connection to the database specifically to handle the transaction. Take this into account when calculating the number of data source connections.

On UNIX platforms, a separate DB2 process is created for each connection and these processes quickly affect performance on systems with low memory and cause errors.

If clones are used, one data pool exists for each clone. This is important when configuring the database maximum connections.

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. 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 a *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.

It is also important to note that when using *connection sharing*, it is only possible to share connections obtained from the same connection pool.

Avoiding a deadlock: Deadlock can occur if the application requires more than one concurrent connection per thread, and the database connection pool is not large enough for the number of threads. Suppose each of the application threads requires two concurrent database connections and the number of threads is equal to the maximum connection pool size. Deadlock can occur when both of the following are true:

- Each thread has its first database connection, and all are in use.
- Each thread is waiting for a second database connection, and none would become available since all threads are blocked.

To prevent the deadlock in this case, the **Maximum Connections** value for the database connection pool should be increased by at least one. Doing this allows for at least one of the waiting threads to obtain its second database connection and to avoid a deadlock.

To avoid deadlock, code the application to use, at most, one connection per thread. If the application is coded to require *C* concurrent database connections per thread, the connection pool must support at least the following number of connections, where *T* is the maximum number of threads.

```
T * (C - 1) + 1
```

The connection pool settings are directly related to the number of connections that the database server is configured to support. If the maximum number of connections in the pool is raised, and the corresponding settings in the database are not raised, the application fails and SQL exception errors are displayed in the `stderr.log` file.

Deferred Enlistment: In the WebSphere Application Server environment, *deferred enlistment* is a term used to refer to the technique of waiting until a connection is first used to enlist it in its unit of work (UOW) scope.

In one example, the technique works like this: a component calls getConnection() from within a global transaction, and at some point later in time, the component uses the connection. The call that uses the connection is intercepted, and the XA resource for that connection is enlisted with the transaction service (which in turn calls XAResource.start()). Next, the actual call is sent to the resource manager.

In contrast, if a component gets a connection within a global transaction without deferred enlistment, then the connection is enlisted in the transaction and has all the overhead associated with that transaction. For XA connections, this includes the two phase commit (2PC) protocol to the resource manager. Deferred enlistment offers better performance in the case where a connection is obtained, but not used within the UOW scope. This saves all the overhead of participating in the UOW when it is not needed.

The WebSphere Application Server relational resource adapter automatically supports deferred enlistment without any additional configuration needed.

Lazy Transaction Enlistment Optimization: The J2EE Connector Architecture (JCA) Version 1.5 specification calls the deferred enlistment technique *lazy transaction enlistment optimization*. This support comes through a marker interface (LazyEnlistableManagedConnection) and a new method on the connection manager (LazyEnlistableConnectionManager()):

```
package javax.resource.spi;
import javax.resource.ResourceException;
import javax.transaction.xa.Xid;

interface LazyEnlistableConnectionManager { // application server
    void lazyEnlist(ManagedConnection) throws ResourceException;
}

interface LazyEnlistableManagedConnection { // resource adapter
}
```

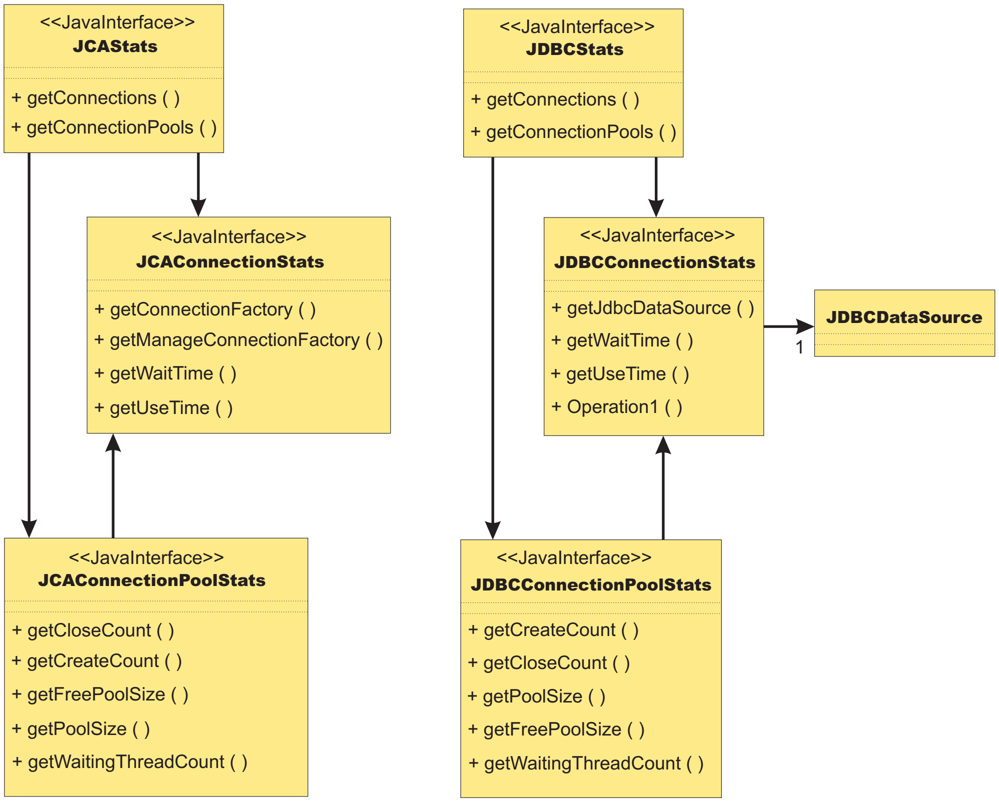
A resource adapter is not required to support this functionality. Check with the resource adapter provider if you need to know if the resource adapter provides this functionality.

Connection and connection pool statistics: Performance Monitoring Infrastructure (PMI) method calls that are supported in the two existing Connection Managers (JDBC and J2C) are still supported in this version of WebSphere Application Server. The calls include:
- ManagedConnectionsCreated
- ManagedConnectionsAllocated
- ManagedConnectionFreed
- ManagedConnectionDestroyed
- BeginWaitForConnection
- EndWaitForConnection
- ConnectionFaults
- Average number of ManagedConnections in the pool
- Percentage of the time that the connection pool is using the maximum number of ManagedConnections
- Average number of threads waiting for a ManagedConnection
- Average percent of the pool that is in use
- Average time spent waiting on a request
- Number of ManagedConnections that are in use
- Number of Connection Handles
- FreePoolSize
- UseTime

Java Specification Request (JSR) 77 requires statistical data to be accessed through managed beans (Mbeans) to facilitate this. The Connection Manager passes the ObjectNames of the Mbeans created for this pool. In the case of Java Message Service (JMS) *null* is passed in. The interface used is :

```
PmiFactory.createJ2CPerf(
    String pmiName, // a unique Identifier for JCA /JDBC.  This is the
                    // ConnectionFactory name.

    ObjectName providerName,// the ObjectName of the J2CResourceAdapter
                            // or JDBCProvider Mbean

    ObjectName factoryName // the ObjectName of the J2CConnectionFactory
                           // or DataSourceMbean.
)
```

The following Unified Modeling Language (UML) diagram shows how JSR 77 requires statistics to be reported :



In WebSphere Application Server Version 5.x, the JCAStats interface was implemented by the J2CResourceAdapter Mbean, and the JDBCStats interface was implemented by the JDBCProvider Mbean. The JCAConnectionStats and JDBCConnectionStats interfaces are not implemented because they collect statistics for 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 1144 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 “Setting client information implicitly” on page 1152.

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.

Setting client information implicitly: You can choose to *explicitly* pass client information of application requests to database connections by calling an IBM proprietary API, setClientInformation(Properties), on the com.ibm.websphere.rsadapter.WSConnection object within your application code. In some cases, however, you might want WebSphere Application Server to handle the passing of client information to database connections. This method of setting the client information is referred to as *implicit*. You might choose the implicit method because:

- You want to keep your application free of proprietary APIs, *or*
- Your application uses container-managed persistence (CMP), in which case you cannot use the proprietary API to set client information on database connections.

The WebSphere Application Server trace facility provides the capability for setting client information implicitly. You can designate one of two special trace groups to enable or disable client information passing: `WAS.clientinfo` trace or `WAS.clientinfopluslogging` trace .

Note:

Connection sharing:

In the case of connection sharing, setting the client information implicitly is done only on the first acquired connection handle. If connection sharing is enabled and two or more getConnection() methods are called (resulting in two handles on the same connection), then only the first getConnection() call causes the client information to be implicitly passed to the back-end database. This is not true for the explicit case; every setClientinformation() method is driven down to the database regardless of connection sharing.

Implicit/explicit co-existence:

When both explicit and implicit are used, some combination of the explicitly set data and implicitly set data is combined, but for the most part the explicit 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:

```
000325_WSRdbManagedConnectionImpl@1234_myAccountingInfo: where 000325 is the
 thread id, WSRdbManagedConnectionImpl@1234 is the websphere connection instance.
```

Client information reset:

In the implicit case, client information is reset when the connection is put back in the pool and only if the WAS.clientinfo and WAS.clientinfopluslogging are disabled (that is, WAS.clientinfo=all=disabled:WAS.clientinfopluslogging=all=disabled).

In the explicit case however, the reset is done only when the application issues setClientInformation(null) on the WSConnection.

WAS.clientinfo trace

By default, the implicit mechanism is disabled. You can turn on this mechanism dynamically (without stopping and starting your application server) or statically by setting the WebSphere Application Server trace group *WAS.clientinfo=all=enabled*.

The information implicitly collected and set on the database connection consists of the *user name*, *user location* and *application name*.

Note: User name and user location can only be implicitly collected and set on the database connection if you enable WebSphere global security.

username

Name of the user that initiated 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 *WSSecurityHelper.getFirstCaller()*.

user location

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 *WSSecurityHelper.getFirstServer().*

application name

Name of the application running. This value is the output of *getApplication()* 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 RRA trace, and 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, 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 were 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 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″ in the information center 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^{TM} 2 Platform, Enterprise Edition (J2EE^{TM})
- Java^{TM} Management Extensions (JMX)
- JDBC^{TM} 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^{TM} 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^{TM} 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^{TM} 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

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' databases requirements.
2. Migrate Version 4.0 data access applications
3. 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 1164.
4. 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.
5. If your enterprise application contains an application client that accesses a relational database, see Configuring data access for application clients.
6. 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.

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. The container-managed bean (CMP) enterprise bean JAR file has a *Table.ddl* file that the assembly tool generates. Extract the *Table.ddl* file to a working directory such as *C:\temp*.
2. Run this command -- **C:\temp>db2cmd** A new command window appears in which you enter the following commands.
3. Run this command -- **C:\temp>db2 connect to *dbname***
4. Run this command -- **C:\temp>db2 -tf Table.ddl** The command runs and creates tables for your CMP enterprise bean.
5. Run this command -- **C:\temp>db2 disconnect all**

   Note: If you use DB2 on Unix, use these same commands. Simply run them from a user with permissions for DB2, rather than from a DB2 command window.

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″ in the information center.
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″ in the information center.
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 1162 and “WebSphere relational resource adapter settings” on page 1128.
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

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; see the topic Data Sources (Version 4). If you are using more advanced releases of these specifications, see the topic Data Source collection .

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″ in the information center.

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″ in the information center. (For V4 data sources, see ″Configuring new WAS40 data sources using scripting.″)

   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 the *resource.xml* file. You should not need to modify the **jdbc-resource-provider-templates.xml**. However, special consideration should be taken if you need to update the **jdbc-resource-provider-templates.xml file**. For details, consult J2EE Connector Architecture migration tips.

Verifying a connection:

Many connection problems can be easily fixed by verifying some configuration parameters. This article provides a checklist of steps that you must complete to enable a successful connection. Click on the link for more information on a specific step.

If your connection is still not successful after completing these steps and reviewing the applicable information, check the SystemOut.log for warning or exception messages. Then use the technical support search function to find known problems.

1. Create the authentication data alias.
2. Create the JDBC provider.
3. Create a data source.
4. Save the data source.
5. If you created a new authentication alias, restart the server for which you need to verify connectivity.
6. Test the connection

   You can test your connection from the data source collection view or the data source details view. Access either view in the administrative console, and then select a connection from the list. Click the **Test Connection** button on the connection.

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″ in the information center.

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 1135 topic.)
11. Click **OK** to return to the JDBC providers page, where your new JDBC provider appears in the list.

For more information about creating a data source for association with your JDBC provider, see "Creating and configuring a data source using the administrative console" on page 1171.

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*.

Name:

Specifies the name of the resource provider.

Data type	String

Description:

Specifies a text description for the resource provider.

Data type	String

Class path:

Specifies a list of paths or JAR file names which together form the location for the resource provider classes.

For example, *D:/SQLLIB/java/db2java.zip*.

Class path entries are separated by using the ENTER key and must not contain path separator characters (such as ';' or ':'). Class paths contain variable (symbolic) names which you can substitute using a variable map. Check the driver installation notes for the specific required JAR file names.

Data type	String

Native Library Path:

Specifies a list of paths that forms the location for the resource provider native libraries.

Native path entries are separated by using the ENTER key and must not contain path separator characters (such as ';' or ':'). Native paths can contain variable (symbolic) names which you can substitute using a variable map.

Data type	String

Implementation class name:

Specifies the Java class name of the JDBC driver implementation.

This class is available in the driver file mentioned in the class path description above. For example, *COM.ibm.db2.jdbc.DB2XADataSource*.

Note: If you modify the implementation class name of the JDBC provider after you have created the provider, you might disconnect the provider from the template used to create it. As a result, data sources created from this JDBC provider do not have an associated template; you must manually configure a working data source through setting custom properties.

Data type	String

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	
	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	Recommended for use with WebSphere Application Server run on Windows, UNIX, or workstation-based LINUX
	DB2 UDB for iSeries (Toolbox XA)	One and two phase	Recommended for use with WebSphere Application Server run on Windows, UNIX, or workstation-based LINUX
	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	*Only* for use with WebSphere Application Server run on Windows, UNIX, or workstation-based LINUX
	DB2 Universal JDBC Provider (XA)	One and two phase	*Only* for use with WebSphere Application Server run on Windows, UNIX, or workstation-based LINUX
	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	
Oracle	Oracle JDBC Driver	One phase only	
	Oracle JDBC Driver (XA)	One and two phase	

Database type	JDBC Provider	Transaction support	Version and other considerations
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

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.

You can also use the Tivoli Performance Viewer to minimize cache discards. Use a standard workload that represents a typical number of incoming client requests, use a fixed number of iterations, and use a standard set of configuration settings. **Note:** The higher the statement cache, the more system resources are delayed. Therefore, if you set the number too high, you could lack resources because your system is not able to open that many prepared statements.

In test applications, tuning the statement cache improved throughput by 10-20%. However, because of potential resource limitations, this might not always be possible.

Data type	Integer
Default	Depends on the database. Most are 10. Informix 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 Javadoc 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 collection:

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” 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 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 1220.

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. Set 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 (shown on 2 lines for publication):

```
set classpath=%classpath%;D:\WebSphere\AppServer\lib\wsexception.jar;
  D:\WebSphere\AppServer\lib\wasjmx.jar
```

2. Look up the host and get an administration client handle.
3. Get a configuration service handle.
4. Update the *resource.xml* file using the configuration service as desired.
   a. Add a JDBC provider.
   b. Add the data source.
   c. Add the connection factory. (This step is necessary only for data sources that must support container-managed persistence.)
5. Reload the *resource.xml* file to bind the newly created data source into the JNDI namespace. Perform this step if you want to use the newly created data source right away without restarting the application server.
   a. Locate the DataSourceConfigHelper MBean using the name.
   b. Put together the signature and parameters for the call.
   c. Invoke the reload() call.

Example: Using the Java Management Extensions API to create a JDBC driver and data source for container-managed persistence:

```
//
// "This program may be used, executed, copied, modified and distributed without royalty for the
// purpose of developing, using, marketing, or distributing."
//
// Product 5630-A36,  (C) COPYRIGHT International Business Machines Corp., 2001, 2002
// All Rights Reserved * Licensed Materials - Property of IBM
//
import java.util.*;
import javax.sql.*;
import javax.transaction.*;
import javax.management.*;

import com.ibm.websphere.management.*;
import com.ibm.websphere.management.configservice.*;
import com.ibm.ws.exception.WsException;


/**
 * Creates a node scoped resource.xml entry for a DB2 XA datasource.
 * The datasource created is for CMP use.
 *
 * We need following to run (shown on multiple lines for publication):
 * set classpath=%classpath%;D:\WebSphere\AppServer\lib\wsexception.jar;
                D:\WebSphere\AppServer\lib\wasjmx.jar;D:\$WAS_HOME\lib\wasx.jar
  */
public class CreateDataSourceCMP {

   String dsName = "markSection"; // ds display name , also jndi name and CF name
   String dbName = "SECTION";     // database name
   String authDataAlias = "db2admin"; // an authentication data alias
   String uid = "db2admin";           // userid
   String pw  = "db2admin";           //  password
   String dbclasspath = "D:/SQLLIB/java/db2java.zip"; // path to the db driver


   /**
    * Main method.
    */
   public static void main(String[] args) {
      CreateDataSourceCMP cds = new CreateDataSourceCMP();
      try {
         cds.run(args);
      } catch (com.ibm.ws.exception.WsException ex) {
         System.out.println("Caught this " + ex   );
         ex.printStackTrace();
         //ex.getCause().printStackTrace();
      } catch (Exception ex) {
         System.out.println("Caught this " + ex );
         ex.printStackTrace();
      }
   }


   /**
    * This method creates the datasource using JMX.
* The datasource created here is only written into resources.xml.
    * It is not bound into namespace until the server is restarted, or an application started
    */
   public void run(String[] args) throws Exception {

      try {
         // Initialize the AdminClient.
         Properties adminProps = new Properties();
         adminProps.setProperty(AdminClient.CONNECTOR_TYPE, AdminClient.CONNECTOR_TYPE_SOAP);
         adminProps.setProperty(AdminClient.CONNECTOR_HOST, "localhost");
         adminProps.setProperty(AdminClient.CONNECTOR_PORT, "8880");
         AdminClient adminClient = AdminClientFactory.createAdminClient(adminProps);

         // Get the ConfigService implementation.
         com.ibm.websphere.management.configservice.ConfigServiceProxy configService =
         new com.ibm.websphere.management.configservice.ConfigServiceProxy(adminClient);

         Session session = new Session();

         // Use this group to add to the node scoped resource.xml.
         ObjectName node1 = ConfigServiceHelper.createObjectName(null, "Node", null);
         ObjectName[] matches = configService.queryConfigObjects(session, null, node1, null);
         node1 = matches[0];     // use the first node found

         // Use this group to add to the server1 scoped resource.xml.
         ObjectName server1 = ConfigServiceHelper.createObjectName(null, "Server", "server1");
         matches = configService.queryConfigObjects(session, null, server1, null);
         server1 = matches[0];   // use the first server found

         // Create the JDBCProvider
         String providerName = "DB2 JDBC Provider (XA)";
         System.out.println("Creating JDBCProvider " + providerName );

         // Prepare the attribute list
         AttributeList provAttrs = new AttributeList();
         provAttrs.add(new Attribute("name", providerName));
         provAttrs.add(new Attribute("implementationClassName",
                                "COM.ibm.db2.jdbc.DB2XADataSource"));
         provAttrs.add(new Attribute("description","DB2 JDBC2-compliant XA Driver"));

         //create it
         ObjectName jdbcProv =
           configService.createConfigData(session,node1,"JDBCProvider",
                                     "resources.jdbc:JDBCProvider",provAttrs);
         // now plug in the classpath
configService.addElement(session,jdbcProv,"classpath",dbclasspath,-1);


// Search for RRA so we can link it to the datasource
         ObjectName rra =
           ConfigServiceHelper.createObjectName(null, "J2CResourceAdapter", null);
         matches = configService.queryConfigObjects(session, node1, rra, null);
         rra = matches[0]; // use the first J2CResourceAdapter segment for builtin_rra

         // Prepare the attribute list
         AttributeList dsAttrs = new AttributeList();
         dsAttrs.add(new Attribute("name", dsName));
         dsAttrs.add(new Attribute("jndiName", "jdbc/" + dsName));
         dsAttrs.add(new Attribute("datasourceHelperClassname",
                                   "com.ibm.websphere.rsadapter.DB2DataStoreHelper"));
         dsAttrs.add(new Attribute("statementCacheSize", new Integer(10)));
         dsAttrs.add(new Attribute("relationalResourceAdapter",
            rra)); // this is where we make the link to "builtin_rra"
         dsAttrs.add(new Attribute("description", "JDBC Datasource for mark section CMP 2.0 test"));
         dsAttrs.add(new Attribute("authDataAlias",authDataAlias));

         // Create the datasource
         System.out.println("  **  Creating datasource");
         ObjectName dataSource =
           configService.createConfigData(session,jdbcProv,"DataSource",
           "resources.jdbc:DataSource",dsAttrs);

         // Add a propertySet.
         AttributeList propSetAttrs = new AttributeList();
         ObjectName resourcePropertySet =
           configService.createConfigData(session,dataSource,"propertySet","",propSetAttrs);

         // Add resourceProperty databaseName
         AttributeList propAttrs1 = new AttributeList();
         propAttrs1.add(new Attribute("name", "databaseName"));
         propAttrs1.add(new Attribute("type", "java.lang.String"));
         propAttrs1.add(new Attribute("value", dbName));

         configService.addElement(session,resourcePropertySet,"resourceProperties",propAttrs1,-1);

// Now Create the corresponding J2CResourceAdapter Connection Factory object.
         ObjectName jra = ConfigServiceHelper.createObjectName(null,"J2CResourceAdapter",null);

         // Get all the J2CResourceAdapter, and I want to add my datasource
         System.out.println("  **  Get all J2CResourceAdapter's");
         ObjectName[] jras = configService.queryConfigObjects(session, node1, jra, null);

         int i=0;

         for (;i< jras.length;i++) {
            System.out.println(ConfigServiceHelper.getConfigDataType(jras[i])+ " " + i + " = "
                 + jras[i].getKeyProperty(SystemAttributes._WEBSPHERE_CONFIG_DATA_DISPLAY_NAME)
                     + "\nFrom scope ="
                     + jras[i].getKeyProperty(SystemAttributes._WEBSPHERE_CONFIG_DATA_ID));
            // quit on the first builtin_rra
            if (jras[i].getKeyProperty(SystemAttributes._WEBSPHERE_CONFIG_DATA_DISPLAY_NAME)
                .equals("WebSphere Relational Resource Adapter")) {
               break;
            }
         }


         if (i >= jras.length) {
            System.out.println(
              "Did not find builtin_rra J2CResourceAdapter object creating CF anyways" );
         } else {
            System.out.println(
              "Found builtin_rra J2CResourceAdapter object at index " + i + " creating CF" );
         }

         // Prepare the attribute list
         AttributeList cfAttrs = new AttributeList();
         cfAttrs.add(new Attribute("name", dsName + "_CF"));
```

```
        cfAttrs.add(new Attribute("authMechanismPreference","BASIC_PASSWORD"));
        cfAttrs.add(new Attribute("authDataAlias",authDataAlias));
        cfAttrs.add(new Attribute("cmpDatasource",
            dataSource ));  // this is where we make the link to DataSource's xmi:id
        ObjectName cf =
           configService.createConfigData(session,jras[i],"CMPConnectorFactory",
                                          "resources.jdbc:CMPConnectorFactory",cfAttrs);


        // ===== start Security section
        System.out.println("Creating an authorization data alias " + authDataAlias);

// Find the parent security object
        ObjectName security = ConfigServiceHelper.createObjectName(null, "Security", null);
        ObjectName[] securityName =
          configService.queryConfigObjects(session, null, security, null);
        security=securityName[0];

        // Prepare the attribute list
        AttributeList authDataAttrs = new AttributeList();
        authDataAttrs.add(new Attribute("alias", authDataAlias));
        authDataAttrs.add(new Attribute("userId", uid));
        authDataAttrs.add(new Attribute("password", pw));
        authDataAttrs.add(new Attribute("description","Auto created alias for datasource"));

        //create it
        ObjectName authDataEntry =
           configService.createConfigData(session,security,"authDataEntries",
                                          "JAASAuthData",authDataAttrs);
        // ===== end Security section

        // Save the session
        System.out.println("Saving session" );
        configService.save(session, false);

        // reload resources.xml to bind the new datasource into the name space
        reload(adminClient,true);
     } catch (Exception ex) {
        ex.printStackTrace(System.out);
        throw ex;
     }
  }

  /**
   * Get the DataSourceConfigHelperMbean and call reload() on it
   *
   * @param adminClient
   * @param verbose true - print messages to stdout
   */
  public void reload(AdminClient adminClient,boolean verbose) {
     if (verbose) {
        System.out.println("Finding the Mbean to call reload()");
     }

     // First get the  Mbean
     ObjectName handle = null;
     try {
        ObjectName queryName = new ObjectName("WebSphere:type=DataSourceCfgHelper,*");
        Set s = adminClient.queryNames(queryName, null);
        Iterator iter = s.iterator();
        if (iter.hasNext()) handle = (ObjectName)iter.next();
     } catch (MalformedObjectNameException mone) {
        System.out.println("Check the program variable queryName" + mone);
     } catch (com.ibm.websphere.management.exception.ConnectorException ce) {
        System.out.println("Cannot connect to the application server" + ce);
     }
```

```
      if (verbose) {
         System.out.println("Calling reload()");
      }
      Object result = null;
      try {
         result = adminClient.invoke(handle, "reload", new Object[] {}, new String[] {});
      } catch (MBeanException mbe) {
         if (verbose) {
            System.out.println("\tMbean Exception calling reload" + mbe);
         }
      } catch (InstanceNotFoundException infe) {
         System.out.println("Cannot find reload ");
      } catch (Exception ex) {
         System.out.println("Exception occurred calling reload()"  + ex);
      }

      if (result==null && verbose) {
         System.out.println("OK reload()");
      }
   }

}
```

Example: Using the Java Management Extensions API to create a JDBC driver and data source for bean-managed persistence, session beans, or servlets:

```
//
// "This program may be used, executed, copied, modified and distributed without royalty for the
// purpose of developing, using, marketing, or distributing."
//
// Product 5630-A36,  (C) COPYRIGHT International Business Machines Corp., 2001, 2002
// All Rights Reserved * Licensed Materials - Property of IBM
//
import java.util.*;
import javax.sql.*;
import javax.transaction.*;
import javax.management.*;

import com.ibm.websphere.management.*;
import com.ibm.websphere.management.configservice.*;
import com.ibm.ws.exception.WsException;


/**
 * Creates a node scoped resource.xml entry for a DB2 XA datasource.
 * The datasource created is for BMP use.
 *
 * We need following to run (shown on 2 lines for publication):
 * set classpath=%classpath%;D:\WebSphere\AppServer\lib\wsexception.jar;
     D:\WebSphere\AppServer\lib\wasjmx.jar;D:\$WAS_HOME\lib\wasx.jar
 */
public class CreateDataSourceBMP {

   String dsName = "markSection"; // ds display name , also jndi name and CF name
   String dbName = "SECTION";     // database name
   String authDataAlias = "db2admin"; // an authentication data alias
   String uid = "db2admin";           // userid
   String pw  = "db2admin";           //  password
   String dbclasspath = "D:/SQLLIB/java/db2java.zip"; // path to the db driver




   /**
    * Main method.
    */
   public static void main(String[] args) {
```

```
   CreateDataSourceBMP cds = new CreateDataSourceBMP();

   try {
      cds.run(args);
   } catch (com.ibm.ws.exception.WsException ex) {
      System.out.println("Caught this " + ex   );
      ex.printStackTrace();
      //ex.getCause().printStackTrace();
   } catch (Exception ex) {
      System.out.println("Caught this " + ex );
      ex.printStackTrace();
   }
}


/**
 * This method creates the datasource using JMX.
 *
 * The datasource created here is only written into resources.xml.
 * It is not bound into namespace until the server is restarted, or an application started
 */
public void run(String[] args) throws Exception {

   try {
      // Initialize the AdminClient.
      Properties adminProps = new Properties();
      adminProps.setProperty(AdminClient.CONNECTOR_TYPE, AdminClient.CONNECTOR_TYPE_SOAP);
      adminProps.setProperty(AdminClient.CONNECTOR_HOST, "localhost");
      adminProps.setProperty(AdminClient.CONNECTOR_PORT, "8880");
      AdminClient adminClient = AdminClientFactory.createAdminClient(adminProps);

      // Get the ConfigService implementation.
      com.ibm.websphere.management.configservice.ConfigServiceProxy configService =
      new com.ibm.websphere.management.configservice.ConfigServiceProxy(adminClient);

      Session session = new Session();

      // Use this group to add to the node scoped resource.xml.
      ObjectName node1 = ConfigServiceHelper.createObjectName(null, "Node", null);
      ObjectName[] matches = configService.queryConfigObjects(session, null, node1, null);
      node1 = matches[0];     // use the first node found

      // Use this group to add to the server1 scoped resource.xml.
      ObjectName server1 = ConfigServiceHelper.createObjectName(null, "Server", "server1");
      matches = configService.queryConfigObjects(session, null, server1, null);
      server1 = matches[0];   // use the first server found

      // Create the JDBCProvider
      String providerName = "DB2 JDBC Provider (XA)";
      System.out.println("Creating JDBCProvider " + providerName );

      // Prepare the attribute list
      AttributeList provAttrs = new AttributeList();
      provAttrs.add(new Attribute("name", providerName));
      provAttrs.add(
        new Attribute("implementationClassName", "COM.ibm.db2.jdbc.DB2XADataSource"));
      provAttrs.add(new Attribute("description","DB2 JDBC2-compliant XA Driver"));

      //create it
      ObjectName jdbcProv =
         configService.createConfigData(session,node1,"JDBCProvider",
         "resources.jdbc:JDBCProvider",provAttrs);
      // now plug in the classpath
      configService.addElement(session,jdbcProv,"classpath",dbclasspath,-1);
```

```
// Search for RRA so we can link it to the datasource
ObjectName rra = ConfigServiceHelper.createObjectName(null, "J2CResourceAdapter", null);
matches = configService.queryConfigObjects(session, node1, rra, null);
rra = matches[0]; // use the first J2CResourceAdapter segment for builtin_rra

// Prepare the attribute list
AttributeList dsAttrs = new AttributeList();
dsAttrs.add(new Attribute("name", dsName));
dsAttrs.add(new Attribute("jndiName", "jdbc/" + dsName));
dsAttrs.add(new Attribute("datasourceHelperClassname",
                          "com.ibm.websphere.rsadapter.DB2DataStoreHelper"));
dsAttrs.add(new Attribute("statementCacheSize", new Integer(10)));
dsAttrs.add(new Attribute(
  "relationalResourceAdapter", rra)); // this is where we make the link to "builtin_rra"
dsAttrs.add(new Attribute("description", "JDBC Datasource for mark section CMP 2.0 test"));
dsAttrs.add(new Attribute("authDataAlias",authDataAlias));

// Create the datasource
System.out.println("  **  Creating datasource");
ObjectName dataSource =
    configService.createConfigData(session,jdbcProv,"DataSource",
    "resources.jdbc:DataSource",dsAttrs);

// Add a propertySet.
AttributeList propSetAttrs = new AttributeList();
ObjectName resourcePropertySet =
   configService.createConfigData(session,dataSource,"propertySet","",propSetAttrs);

// Add resourceProperty databaseName
AttributeList propAttrs1 = new AttributeList();
propAttrs1.add(new Attribute("name", "databaseName"));
propAttrs1.add(new Attribute("type", "java.lang.String"));
propAttrs1.add(new Attribute("value", dbName));

configService.addElement(session,resourcePropertySet,"resourceProperties",propAttrs1,-1);


// ===== start Security section
System.out.println("Creating an authorization data alias " + authDataAlias);

// Find the parent security object
ObjectName security = ConfigServiceHelper.createObjectName(null, "Security", null);
ObjectName[] securityName =
   configService.queryConfigObjects(session, null, security, null);
security=securityName[0];

// Prepare the attribute list
AttributeList authDataAttrs = new AttributeList();
authDataAttrs.add(new Attribute("alias", authDataAlias));
authDataAttrs.add(new Attribute("userId", uid));
authDataAttrs.add(new Attribute("password", pw));
authDataAttrs.add(new Attribute("description","Auto created alias for datasource"));

//create it
ObjectName authDataEntry =
   configService.createConfigData(session,security,"authDataEntries",
   "JAASAuthData",authDataAttrs);
// ===== end Security section

// Save the session
System.out.println("Saving session" );
configService.save(session, false);

// reload resources.xml
reload(adminClient,true);
```

```
      } catch (Exception ex) {
         ex.printStackTrace(System.out);
         throw ex;
      }
   }

   /**
    * Get the DataSourceConfigHelperMbean and call reload() on it
    *
    * @param adminClient
    * @param verbose true - print messages to stdout
    */
   public void reload(AdminClient adminClient,boolean verbose) {
      if (verbose) {
         System.out.println("Finding the Mbean to call reload()");
      }

      // First get the  Mbean
      ObjectName handle = null;
      try {
         ObjectName queryName = new ObjectName("WebSphere:type=DataSourceCfgHelper,*");
         Set s = adminClient.queryNames(queryName, null);
         Iterator iter = s.iterator();
         if (iter.hasNext()) handle = (ObjectName)iter.next();
      } catch (MalformedObjectNameException mone) {
         System.out.println("Check the program variable queryName" + mone);
      } catch (com.ibm.websphere.management.exception.ConnectorException ce) {
         System.out.println("Cannot connect to the application server" + ce);
      }

      if (verbose) {
         System.out.println("Calling reload()");
      }
      Object result = null;
      try {
         result = adminClient.invoke(handle, "reload", new Object[] {}, new String[] {});
      } catch (MBeanException mbe) {
         if (verbose) {
            System.out.println("\tMbean Exception calling reload" + mbe);
         }
      } catch (InstanceNotFoundException infe) {
         System.out.println("Cannot find reload ");
      } catch (Exception ex) {
         System.out.println("Exception occurred calling reload()"  + ex);
      }

      if (result==null && verbose) {
         System.out.println("OK reload()");
      }
   }
}
```

Example: Creating a JDBC provider and data source using Java Management Extensions API and the scripting tool: The following code is a JACL (WSadmin - scripting tool) script used to create a data source.

Note: Use this script to create only data sources for which the product does *not* provide a template. For every JDBC provider WebSphere Application Server supports, the product provides a corresponding data source template. See the topic ″Creating configuration objects using the wsadmin tool″ in the information center 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 1220.

This script:

- 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*

```
#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
```

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 WebSphere variables, see the ″Configuring WebSphere variables″ topic to verify that you define them correctly. A `variable cannot be found` exception results from attempted use of a data source that is configured with undefined variables.

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″ in the information center as a guide for this technique.

Java stand-alone program

Finally, you can test a connection by executing the `testConnection()` method on the `DataSourceCfgHelper` MBean. This method allows you to pass the configuration ID of the configured data source. The Java program connects to a running Java Management Extensions (JMX) server to access the MBean. In a base installation of Application Server, you connect to the JMX server running in the application server, usually on port 8880.

The return value from this invocation is either 0, a positive number, or an exception. 0 indicates that the operation completed successfully, with no warnings. A positive number indicates that the operation completed successfully, with the number of warnings. An exception indicates that the test of the connection failed.

You can find an example of this code in Example: Test a connection using testConnection(ConfigID).

Testing a connection with the administrative console:

After you have defined and saved a data source, you can click the **Test Connection** button to ensure that the parameters in the data source definition are correct. 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 class path or other fields, such as *${DB2_JDBC_DRIVER_PATH}/db2java.zip*, make sure that you assign it a value in the *WebSphere Variables* page. Note that if you add a new WebSphere environment variable, or modify it , the process (application server) might need restarting.
4. 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.

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`.

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.

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 RAID.

For more information about using AIX with DB2 see ″Tuning operating systems″ in the information center.

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.

Use TCP/IP sockets for DB2 on Linux

- **Description:** On Linux platforms, whether the DB2 server resides on a local machine with WebSphere Application Server or on a remote machine, configure the DB2 application databases to use TCP/IP sockets for communications with the database.

- **How to view or set:** Locate the directions for configuring DB2 on Linux in the *Installing your application serving environment* PDF. This document specifies setting DB2COMM for TCP/IP and corresponding changes required in the `etc/service` file.
- **Default value:** Shared memory for local databases
- **Recommended value:** On Linux, change the specification for the DB2 application databases and any session databases from shared memory to TCP/IP sockets.

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 = (# of connections set for data source + # of connections in session manager) x # of clones.

After calculating the MaxAppls settings for the WebSphere 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 was 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 is executed in DB2, various methods are used to calculate the most efficient access plan. The range is from zero 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 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 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 issue runstats on all user and system tables for the database to which you are currently connected. Rebind packages using the **bind** command. If runstats exists, issue the **db2 -v ″select tbname, nleaf, nlevels, stats_time from sysibm.sysindexes″** command on DB2 CLP. If no runstats exist, nleaf and nlevels are -1, and stats_time has an empty entry , for example ″-″. If runstats was previously done, the real-time stamp from completion of the runstats also displays under stats_time. If you think the time shown for the previous runstats is too old, execute runstats again.
- **Default value:** None
- **Recommended value:** None

DB2 MinCommit

- **Description:** Delays the writing of log records to a disk until a minimum number of commits is performed, reducing the database manager overhead associated with writing log records. For example, if MinCommit is set to 2, a second commit causes output to the transaction log for the first and second commits. The exception occurs when a one-second timeout forces the first commit to be output if a second commit does not occur within one second.

  In test applications, up to 90% of the disk input and output was related to the DB2 transaction log. Changing MinCommit from 1 to 2 reduced the results to 45%.

  Adjust this parameter if the disk input and output wait is more than 5% and there is DB2 transaction log activity from multiple sources. When a lot of activity occurs from multiple sources, it is less likely that a single commit has to wait for another commit, or the one second timeout. Do not adjust this parameter if you have an application with a single thread performing a series of commits because each commit can hit the one second delay.
- **How to view or set:**

1. Issue the DB2 **get db cfg for *xxxxxx***command, where *xxxxxx* is the name of the application database, to list database configuration parameters.
2. Look for ″Group commit count (MINCOMMIT)″.
3. Set a new value by issuing the DB2 **update db cfg for *xxxxxx* using mincommit *n*** command, where *xxxxxx* is the name of the application database and *n* is a value between 1 and 25 inclusive.

The new setting takes effect immediately.

The following are several metrics that are related to DB2 MinCommit:

– The disk input and output wait can be observed on AIX with the command **vmstat 5**. This shows statistics every 5 seconds. Look for the *wa* column under the CPU area.

– The percentage of time a disk is active can be observed on AIX with the command **iostat 5**. This shows statistics every 5 seconds. Look for the *%tm_act* column.

– The DB2 **get snapshot for db on *xxxxxx*** command , where *xxxxxx* is the name of the application database, shows counters for log pages read and log pages written.

- **Default value:** 1.
- **Recommended value:** 1 or 2, if the circumstance permits.

DB2 Deadlock Event Monitor

- **Description:**For DB2 V8 or later, deadlock event monitor is turned on by default when a database is created. This event monitor captures critical information about all connections involved in deadlock when the particular event occurs. Although there is not much overhead on faster SMP systems, it can be turned off.
- **How to view or set:** In DB2 command window, issue following commands:

```
db2 connect to [db name]
db2 set event monitor db2detaildeadlock state 0
db2 drop event monitor db2detaildeadlock
db2 connect reset
db2 terminate
```

.

- **Default value:** Event monitor is ON by default.
- **Recommended value:** Turn OFF event monitor, if it is not required.

For more information about using AIX with DB2 see ″Tuning operating systems.″.

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. A list that contains detailed descriptions for all of these providers follows the table.

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	
	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	Recommended for use with WebSphere Application Server run on Windows, UNIX, or workstation-based LINUX
	DB2 UDB for iSeries (Toolbox XA)	One and two phase	Recommended for use with WebSphere Application Server run on Windows, UNIX, or workstation-based LINUX
	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	*Only* for use with WebSphere Application Server run on Windows, UNIX, or workstation-based LINUX
	DB2 Universal JDBC Provider (XA)	One and two phase	*Only* for use with WebSphere Application Server run on Windows, UNIX, or workstation-based LINUX
	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	
Oracle	Oracle JDBC Driver	One phase only	
	Oracle JDBC Driver (XA)	One and two phase	

Database type	JDBC Provider	Transaction support	Version and other considerations
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 JDBC provider list

The following list contains a description of every JDBC provider that can be defined for use with WebSphere Application Server Version 6.x. It also shows the supported data source classes and their required properties. 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 properties list. Rather, inclusion in the list means that a value is typically required for that field.

Use these links to find your provider information:
- DB2 on Windows, UNIX, or workstation-based LINUX
- DB2 UDB for iSeries
- DB2 on z/OS, connecting to Application Server on Windows, UNIX, or workstation-based LINUX
- Cloudscape
- Informix
- Sybase
- Oracle
- MS SQL Server

DB2 on Windows, UNIX, or workstation-based LINUX

1. **DB2 Universal JDBC Provider**

   The DB2 Universal JDBC Driver is an architecture-neutral JDBC driver for distributed and local DB2 access. Because the Universal Driver architecture is independent of any particular JDBC driver connectivity or target platform, it allows both Java connectivity (Type 4) or Java Native Interface (JNI) based connectivity (Type 2) in a single driver instance to DB2.

   This JDBC driver allows applications to use both JDBC and Structured Query Language in Java (SQLJ) access.

   The DB2 Universal JDBC Driver Provider supports one phase data source:

   ```
   com.ibm.db2.jcc.DB2ConnectionPoolDataSource
   ```

   Requires JDBC driver files:
   - `db2jcc.jar` After you install DB2, you can find this jar file in the DB2 *java* directory. For Type 4 JDBC driver support from a client machine where DB2 is not installed, copy this file to the local machine. If you install any fixes or upgrades to DB2, you must update this file as well. You must

also set the ***DB2UNIVERSAL_JDBC_DRIVER_PATH*** path variable to point to the **db2jcc.jar** file. See the Cloudscape section for more information on the ***DB2UNIVERSAL_JDBC_DRIVER_PATH*** path variable.

Note: To find out the version of the universal driver you are using, issue this DB2 command:

```
java com.ibm.db2.jcc.DB2Jcc -version
```

The output for the above example is:

```
IBM DB2 JDBC Universal Driver Architecture 2.2.xx
```

- **db2jcc_license_cu.jar** This is the DB2 Universal JDBC driver license file that allows access to the DB2 Universal database. Use this jar file or the next one to gain access to the database. This jar file ships with WebSphere Application Server in a directory defined by **${UNIVERSAL_JDBC_DRIVER_PATH}**environment variable.
- **db2jcc_license_cisuz.jar** This is the DB2 Universal JDBC driver license file that allows access to the following databases:
  - DB2 Universal
  - DB2 for iSeries
  - DB2 for z/OS
  - SQLDS

  The **db2jcc_license_cisuz.jar** does not ship with Websphere Application Server and should be located in the same directory as the db2jcc.jar file, so that the ***DB2UNIVERSAL_JDBC_DRIVER_PATH*** points to both.

```
The classpath for this provider is set as follows:

   <classpath>${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.jar </classpath>
   <classpath>${UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cu.jar</classpath>
   <classpath>${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cisuz.jar</classpath>
```

Note: The license jar files are independent of each other; therefore, order does not matter.

Requires **DataStoreHelper** class:

```
com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper
```

Requires a valid authentication alias.

Requires properties:

- **databaseName** This is an actual database name if the **driverType** is set to **4**, or a locally cataloged database name if the **driverType** is set to **2**.
- **driverType** The JDBC connectivity type of a data source. *There are two permitted values: 2 and 4*. If you want to use Universal JDBC Type 2 driver, set this value to 2. If you want to use Universal JDBC Type 4 driver, set this value to 4.
- **serverName** The TCP/IP address or host name for the Distributed Relational Database Architecture (DRDA) server. Provide a value for this property only if your **driverType** is set to **4**. This property is not required if your **driverType** is set to **2**.
- **portNumber** The TCP/IP port number where the DRDA server resides. Provide a value for this property only if your **driverType** is set to **4**. This property is not required if your **driverType** is set to **2**.

2. **DB2 Universal JDBC Provider (XA)**

   The DB2 Universal JDBC Provider (XA) is an architecture-neutral JDBC provider for distributed and local DB2 access. Whether you use this provider for Java connectivity or Java Native Interface (JNI) based connectivity depends on the version of DB2 you are running. Application Server Version 6.0 minimally requires DB2 8.1 Fix Pack 6. This version of DB2 only supports XA connectivity over the Java Native Interface (JNI) based connectivity (Type 2) driver. In order to use XA connectivity with the Type 4 driver, DB2 8.1 Fix Pack 7 or higher is required.

   The DB2 Universal JDBC Driver (XA) supports two phase transactions and the more advanced data source option offered by Application Server (as opposed to the other option, Version 4 data sources). This driver also allows applications to use both JDBC and SQLJ access.

The DB2 Universal JDBC Driver Provider supports the two phase data source:

`com.ibm.db2.jcc.DB2XADataSource`

Requires JDBC driver files:

- **`db2jcc.jar`** After you install DB2, you can find this `.jar` file in the DB2 java directory. For Type 4 JDBC driver support from a client machine where DB2 is not installed, copy this file to the local machine. If you install any fixes or upgrades to DB2, you must update this file as well. You must also set the ***DB2UNIVERSAL_JDBC_DRIVER_PATH*** environment variable to point to the `db2jcc.jar` file. See the Cloudscape section for more information on the ***DB2UNIVERSAL_JDBC_DRIVER_PATH*** environment variable.

  Note: To find the level of universal driver you are using, issue the following DB2 command:

  ```
  java com.ibm.db2.jcc.DB2Jcc -version
  ```

  example output of the above:

  ```
  IBM DB2 JDBC Universal Driver Architecture 2.2.xx
  ```
- **`db2jcc_license_cu.jar`** This is the DB2 Universal JDBC driver license file that allows access to the DB2 Universal database. Use this jar file or the next one to gain access to the database. This jar file ships with WebSphere Application Server in the *WAS_HOME/universalDriver/lib* directory.
- **`db2jcc_license_cisuz.jar`** This is the DB2 Universal JDBC driver license file that allows access to the following databases:
  - DB2 Universal
  - DB2 for iSeries
  - DB2 for z/OS
  - SQLDS

  You must use the right license jar file to access a specific database backend.

Requires **`DataStoreHelper`** class:

`com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper`

Requires a valid authentication alias.

Requires properties:

- **databaseName** This is an actual database name if the **driverType** is set to **4**, or a locally cataloged database name if the **driverType** is set to **2**.
- **driverType** The JDBC connectivity type of a data source. *There are two permitted values: 2 and 4*. If you want to use Universal JDBC Type 2 XA driver, set this value to 2. If you want to use Universal JDBC Type 4 XA driver (which requires DB2 8.1 Fix Pack 7 or higher), set this value to 4.
- **serverName** The TCP/IP address or host name for the Distributed Relational Database Architecture (DRDA) server. Provide a value for this property only if your **driverType** is set to **4**. This property is not required if your **driverType** is set to **2**.
- **portNumber** The TCP/IP port number where the DRDA server resides. Provide a value for this property only if your **driverType** is set to **4**. This property is not required if your **driverType** is set to **2**.

3. **DB2 legacy CLI-based Type 2 JDBC Driver**

   The DB2 legacy CLI-based Type 2 JDBC Driver Provider is built on top of DB2 CLI (Call Level Interface). It uses the DB2 CLI interface to communicate with DB2 UDB servers.

   DB2 legacy CLI-based Type 2 JDBC Driver supports one phase data source:

   `COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource`

   Requires JDBC driver files: **`db2java.zip`** (Note: If you run SQLJ in DB2 Version 8, **`db2jcc.jar`** is also required.)

   Requires **`DataStoreHelper`** class:

   `com.ibm.websphere.rsadapter.DB2DataStoreHelper`

   Does not require a valid authentication alias if Application Server is running on the same machine as the database. Otherwise, connectivity through this driver does require an alias.

   Requires properties:

- **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

4. **DB2 legacy CLI-based Type 2 JDBC Driver (XA)**

   The DB2 legacy CLI-based Type 2 JDBC Driver (XA) is built on top of DB2 CLI (Call Level Interface). It uses the DB2 CLI interface to communicate with DB2 UDB servers.

   DB2 legacy CLI-based Type 2 JDBC Driver (XA) supports two phase data source:

   `COM.ibm.db2.jdbc.DB2XADataSource`

   Requires JDBC driver files: **`db2java.zip`** (Note: If you run SQLJ in DB2 Version 8, **`db2jcc.jar`** is also required.)

   Requires **`DataStoreHelper`** class:

   `com.ibm.websphere.rsadapter.DB2DataStoreHelper`

   Does not require a valid authentication alias if Application Server is running on the same machine as the database. Otherwise, connectivity through this driver does require an alias.

   Requires properties:

   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

For more information on DB2, visit the DB2 Web site at: http://www.ibm.com/software/data/db2/.

For information on configuring WebSphere Application Server for DB2 access, see the ″Configuring DB2″ article in the information center.

DB2 UDB for iSeries

1. **DB2 UDB for iSeries (Native)**

   The iSeries Developer Kit for Java contains this Type 2 JDBC driver that is built on top of the iSeries DB2 Call Level Interface (CLI) native libraries. Only use this driver for local DB2 connections on iSeries. It is not recommended for remote access. Use this driver for iSeries V5R2, or later releases.

   DB2 UDB for iSeries (Native V5R2 and later) supports one phase data source:

   `com.ibm.db2.jdbc.app.UDBConnectionPoolDataSource`

   Requires JDBC driver files: **`db2_classes.jar`**

   Requires **`DataStoreHelper`** class:

   `com.ibm.websphere.rsadapter.DB2AS400DataStoreHelper`

   Does not require an authentication alias.

   Requires properties:

   - **databaseName** The name of the relational database to which the data source connections are established. This name must appear in the iSeries Relational Database Directory. The default is *LOCAL.

2. **DB2 UDB for iSeries (Native XA)**

   The iSeries Developer Kit for Java contains this XA-compliant Type 2 JDBC driver built on top of the iSeries DB2 Call Level Interface (CLI) native libraries. Only use this driver for local DB2 connections on iSeries. It is not recommended for remote access. Use this driver for iSeries V5R2 or later releases.

   DB2 UDB for iSeries (Native XA - V5R2 and later) supports two phase data source:

   `com.ibm.db2.jdbc.app.UDBXADataSource`

   Requires JDBC driver files: **`db2_classes.jar`**

   Requires **`DataStoreHelper`** class:

   `com.ibm.websphere.rsadapter.DB2AS400DataStoreHelper`

   Does not require an authentication alias.

   Requires properties:

- **databaseName** The name of the relational database to which the data source connections are established. This name must appear in the iSeries Relational Database Directory. The default is *LOCAL.

3. **DB2 UDB for iSeries (Toolbox)**

   This JDBC driver, also known as iSeries Toolbox driver for Java, is provided in the DB2 for iSeries database server. Use this driver for remote DB2 connections on iSeries. We recommend you use this driver instead of the IBM Developer Kit for Java JDBC Driver to access remote DB2 UDB for iSeries systems.

   DB2 UDB for iSeries (Toolbox) supports one phase data source:

   `com.ibm.as400.access.AS400JDBCConnectionPoolDataSource`

   Requires JDBC driver files: **`jt400.jar`**

   Requires **`DataStoreHelper`** class:

   `com.ibm.websphere.rsadapter.DB2AS400DataStoreHelper`

   Does not require an authentication alias if WebSphere Application Server and DB2 UDB for iSeries are installed in the same server. If they are installed in different servers, the user ID and password are required.

   Requires properties:
   - **serverName** The name of the server from which the data source obtains connections. Example: *myserver.mydomain.com*.

4. **DB2 UDB for iSeries (Toolbox XA)**

   This XA compliant JDBC driver, also known as iSeries Toolbox XA compliant driver for Java, is provided in the DB2 for iSeries database server. Use this driver for remote DB2 connections on iSeries. We recommend you use this driver instead of the IBM Developer Kit for Java JDBC Driver to access remote DB2 UDB for iSeries systems.

   DB2 UDB for iSeries (Toolbox XA) supports two phase data source:

   `com.ibm.as400.access.AS400JDBCXADataSource`

   Requires JDBC driver files: **`jt400.jar`**

   Requires **`DataStoreHelper`** class:

   `com.ibm.websphere.rsadapter.DB2AS400DataStoreHelper`

   Does not require an authentication alias if WebSphere Application Server and DB2 UDB for iSeries are installed in the same server. If they are installed in different servers, the user ID and password are required.

   Requires properties:
   - **serverName** The name of the server from which the data source obtains connections. Example: *myserver.mydomain.com*.

5. **DB2 legacy CLI-based Type 2 JDBC Driver**

   The DB2 legacy CLI-based Type 2 JDBC Driver Provider is built on top of DB2 CLI (Call Level Interface). It uses the DB2 CLI interface to communicate with DB2 UDB servers. This provider is intended for *remote* connections to DB2 running on iSeries; for use with Application Server on Windows, UNIX, or workstation-based LINUX, it therefore requires the DB2 Connect Driver (which is available from DB2).

   DB2 legacy CLI-based Type 2 JDBC Driver supports one phase data source:

   `COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource`

   Requires JDBC driver files: **`db2java.zip`** (Note: If you run SQLJ in DB2 Version 8, **`db2jcc.jar`** is also required.)

   Requires **`DataStoreHelper`** class:

   `com.ibm.websphere.rsadapter.DB2DataStoreHelper`

   Does not require a valid authentication alias.

   Requires properties:

- **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

6. **DB2 legacy CLI-based Type 2 JDBC Driver (XA)**

   The DB2 legacy CLI-based Type 2 JDBC Driver (XA) is built on top of DB2 CLI (Call Level Interface). It uses the DB2 CLI interface to communicate with DB2 UDB servers. This provider is intended for *remote* connections to DB2 running on iSeries; for use with Application Server on Windows, UNIX, or workstation-based LINUX, it therefore requires the DB2 Connect Driver (which is available from DB2).

   DB2 legacy CLI-based Type 2 JDBC Driver (XA) supports two phase data source:

   `COM.ibm.db2.jdbc.DB2XADataSource`

   Requires JDBC driver files: **`db2java.zip`** (Note: If you run SQLJ in DB2 Version 8, **`db2jcc.jar`** is also required.)

   Requires **`DataStoreHelper`** class:

   `com.ibm.websphere.rsadapter.DB2DataStoreHelper`

   Does not require a valid authentication alias.

   Requires properties:

   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

7. **DB2 UDB for iSeries (Native - Version 5 Release 1 and earlier)** -- Deprecated

   This JDBC provider is deprecated because it corresponds to a version of the iSeries operating system that WebSphere Application Server Version 6.x does not support. You must now use iSeries V5R2 or a later release of the iSeries operating system, for which the WebSphere Application Server administrative console lists one native iSeries DB2 non-XA provider: DB2 UDB for iSeries (Native).

   The iSeries Developer Kit for Java contains this Type 2 JDBC driver that is built on top of the iSeries DB2 Call Level Interface (CLI) native libraries. Only use this driver for local DB2 connections on iSeries. It is not recommended for remote access. Use this driver for iSeries V5R1, or earlier releases.

   DB2 UDB for iSeries (Native V5R1 and earlier) supports one phase data source:

   `com.ibm.db2.jdbc.app.DB2StdConnectionPoolDataSource`

   Requires JDBC driver files: **`db2_classes.jar`**

   Requires **`DataStoreHelper`** class:

   `com.ibm.websphere.rsadapter.DB2AS400DataStoreHelper`

   Does not require an authentication alias.

   Requires properties:

   - **databaseName** The name of the relational database to which the data source connections are established. This name must appear in the iSeries Relational Database Directory. The default is *LOCAL.

8. **DB2 UDB for iSeries (Native XA - Version 5 Release 1 and earlier)** -- Deprecated

   This JDBC provider is deprecated because it corresponds to a version of the iSeries operating system that WebSphere Application Server Version 6.x does not support. You must now use iSeries V5R2 or a later release of the iSeries operating system, for which the administrative console lists one native iSeries DB2 XA provider: DB2 UDB for iSeries (Native XA).

   The iSeries Developer Kit for Java contains this XA-compliant Type 2 JDBC driver built on top of the iSeries DB2 Call Level Interface (CLI) native libraries. Only use this driver for local DB2 connections on iSeries. It is not recommended for remote access. Use this driver for iSeries V5R1, or earlier releases.

   DB2 UDB for iSeries (Native XA - V5R1 and earlier) supports two phase data source:

   `com.ibm.db2.jdbc.app.DB2StdXADataSource`

   Requires JDBC driver files: **`db2_classes.jar`**

   Requires **`DataStoreHelper`** class:

   `com.ibm.websphere.rsadapter.DB2AS400DataStoreHelper`

   Does not require an authentication alias.

   Requires properties:

- **databaseName** The name of the relational database to which the data source connections are established. This name must appear in the iSeries Relational Database Directory. The default is *LOCAL.

For more information on DB2 UDB for iSeries, visit the DB2 Web site at: http://www.ibm.com/software/data/db2/

DB2 on z/OS, connecting to Application Server on Windows, UNIX, or workstation-based LINUX

1. **DB2 Universal JDBC Provider**

   The DB2 Universal JDBC Driver is an architecture-neutral JDBC driver for distributed and local DB2 access. Because the Universal Driver architecture is independent of any particular JDBC driver connectivity or target platform, it allows both Java connectivity (Type 4) or Java Native Interface (JNI) based connectivity (Type 2) in a single driver instance to DB2. Starting with WebSphere Application Server Version 5.0.2, the product now supports both Type 2 and Type 4 JDBC drivers. To use the Type 4 driver, you must install DB2 Version 8.1 or a later version. To use the Type 2 driver, you must install DB2 Version 8.1 Fix Pack 2 or a later version.

   This JDBC driver allows applications to use both JDBC and Structured Query Language in Java (SQLJ) access.

   The DB2 Universal JDBC Driver Provider supports one phase data source:

   `com.ibm.db2.jcc.DB2ConnectionPoolDataSource`

   Requires JDBC driver files:

   - **`db2jcc.jar`** After you install DB2, you can find this jar file in the DB2 *java* directory. For Type 4 JDBC driver support from a client machine where DB2 is not installed, copy this file to the local machine. If you install any fixes or upgrades to DB2, you must update this file as well. You must also set the ***DB2UNIVERSAL_JDBC_DRIVER_PATH*** path variable to point to the **`db2jcc.jar`** file. See the Cloudscape section for more information on the ***DB2UNIVERSAL_JDBC_DRIVER_PATH*** path variable.

     Note: To find out the version of the universal driver you are using, issue this DB2 command:

     ```
     java com.ibm.db2.jcc.DB2Jcc -version
     ```

     The output for the above example is:

     ```
     IBM DB2 JDBC Universal Driver Architecture 2.2.xx
     ```
   - **`db2jcc_license_cu.jar`** This is the DB2 Universal JDBC driver license file that allows access to the DB2 Universal database. Use this jar file or the next one to gain access to the database. This jar file ships with WebSphere Application Server in a directory defined by **${UNIVERSAL_JDBC_DRIVER_PATH}**environment variable.
   - **`db2jcc_license_cisuz.jar`** This is the DB2 Universal JDBC driver license file that allows access to the following databases:
     - DB2 Universal
     - DB2 for iSeries
     - DB2 for z/OS
     - SQLDS

     The **db2jcc_license_cisuz.jar** does not ship with Websphere Application Server and should be located in the same directory as the db2jcc.jar file, so that the ***DB2UNIVERSAL_JDBC_DRIVER_PATH*** points to both.

   ```
   The classpath for this provider is set as follows:

      <classpath>${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.jar </classpath>
      <classpath>${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cu.jar</classpath>
      <classpath>${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cisuz.jar</classpath>
   ```

   Note: The license jar files are independent of each other; therefore, order does not matter.

   Requires **`DataStoreHelper`** class:

`com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper`

Requires a valid authentication alias.

Requires properties:

- **databaseName** This is an actual database name if the **driverType** is set to **4**, or a locally cataloged database name if the **driverType** is set to **2**.
- **driverType** The JDBC connectivity type of a data source. *There are two permitted values: 2 and 4*. If you want to use Universal JDBC Type 2 driver, set this value to 2. If you want to use Universal JDBC Type 4 driver, set this value to 4.
- **serverName** The TCP/IP address or host name for the Distributed Relational Database Architecture (DRDA) server. Provide a value for this property only if your **driverType** is set to **4**. This property is not required if your **driverType** is set to **2**.
- **portNumber** The TCP/IP port number where the DRDA server resides. Provide a value for this property only if your **driverType** is set to **4**. This property is not required if your **driverType** is set to **2**.

2. **DB2 Universal JDBC Provider (XA)**

   The DB2 Universal JDBC Driver (XA) is an architecture-neutral JDBC driver for distributed and local DB2 access. In WebSphere Application Server Version 5.0.2, this driver only supports Java Native Interface (JNI) based connectivity (Type 2) in a single driver instance to DB2. To use this driver, you must install DB2 Version 8.1 Fix Pack 2 or a later version. This driver supports two phase transactions and the WebSphere Application Server Version 5.0 data source. This driver allows applications to use both JDBC and SQLJ access.

   The DB2 Universal JDBC Driver Provider supports the two phase data source:

   `com.ibm.db2.jcc.DB2XADataSource`

   Requires JDBC driver files:

   - **`db2jcc.jar`** After you install DB2, you can find this `.jar` file in the DB2 java directory. For Type 4 JDBC driver support from a client machine where DB2 is not installed, copy this file to the local machine. If you install any fixes or upgrades to DB2, you must update this file as well. You must also set the ***DB2UNIVERSAL_JDBC_DRIVER_PATH*** environment variable to point to the `db2jcc.jar` file. See the Cloudscape section for more information on the ***DB2UNIVERSAL_JDBC_DRIVER_PATH*** environment variable.

     Note: To find the level of universal driver you are using, issue the following DB2 command:

     ```
     java com.ibm.db2.jcc.DB2Jcc -version
     ```

     example output of the above:

     ```
     IBM DB2 JDBC Universal Driver Architecture 2.2.xx
     ```
   - **`db2jcc_license_cu.jar`** This is the DB2 Universal JDBC driver license file that allows access to the DB2 Universal database. Use this jar file or the next one to gain access to the database. This jar file ships with WebSphere Application Server in the *WAS_HOME/universalDriver/lib* directory.
   - **`db2jcc_license_cisuz.jar`** This is the DB2 Universal JDBC driver license file that allows access to the following databases:
     - DB2 Universal
     - DB2 for iSeries
     - DB2 for z/OS
     - SQLDS

     You must use the right license jar file to access a specific database backend.

   Requires **`DataStoreHelper`** class:

   `com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper`

   Requires a valid authentication alias.

   Requires properties:

   - **databaseName** This is a locally cataloged database name.
   - **driverType** This is the JDBC connectivity type of a data source. *If you are running a version of DB2 prior to DB2 V8.1 FP6, you are restricted to using only the type 2 driver.*

- **serverName** The TCP/IP address or host name for the Distributed Relational Database Architecture (DRDA) server. Provide a value for this property only if your **driverType** is set to **4**. This property is not required if your **driverType** is set to **2**.
- **portNumber** The TCP/IP port number where the DRDA server resides. Provide a value for this property only if your **driverType** is set to **4**. This property is not required if your **driverType** is set to **2**.

3. **DB2 legacy CLI-based Type 2 JDBC Driver**

   The DB2 legacy CLI-based Type 2 JDBC Driver Provider is built on top of DB2 CLI (Call Level Interface). It uses the DB2 CLI interface to communicate with DB2 UDB servers. For use with Application Server on Windows, UNIX, or workstation-based LINUX, this provider requires DB2 Connect (which is available from DB2).

   DB2 legacy CLI-based Type 2 JDBC Driver supports one phase data source:

   `COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource`

   Requires JDBC driver files: **`db2java.zip`** (Note: If you run SQLJ in DB2 Version 8, **`db2jcc.jar`** is also required.)

   Requires **`DataStoreHelper`** class:

   `com.ibm.websphere.rsadapter.DB2DataStoreHelper`

   Does not require a valid authentication alias.

   Requires properties:

   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

4. **DB2 legacy CLI-based Type 2 JDBC Driver (XA)**

   The DB2 legacy CLI-based Type 2 JDBC Driver (XA) is built on top of DB2 CLI (Call Level Interface). It uses the DB2 CLI interface to communicate with DB2 UDB servers. For use with Application Server on Windows, UNIX, or workstation-based LINUX, this provider requires DB2 Connect (which is available from DB2).

   DB2 legacy CLI-based Type 2 JDBC Driver (XA) supports two phase data source:

   `COM.ibm.db2.jdbc.DB2XADataSource`

   Requires JDBC driver files: **`db2java.zip`** (Note: If you run SQLJ in DB2 Version 8, **`db2jcc.jar`** is also required.)

   Requires **`DataStoreHelper`** class:

   `com.ibm.websphere.rsadapter.DB2DataStoreHelper`

   Does not require a valid authentication alias.

   Requires properties:

   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

For more information on DB2 for z/OS, visit the DB2 Web site at: http://www.ibm.com/software/data/db2/.

Cloudscape

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 vesions 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. See the DB2 section for more information on the **DB2UNIVERSAL_JDBC_DRIVER_PATH** path variable.

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. **WebSphere embedded ConnectJDBC driver for MS SQL Server**

   WebSphere embedded ConnectJDBC driver for MS SQL Server is a Type 4 JDBC driver that provides JDBC access to the MS SQL Server database. This JDBC driver ships with WebSphere Application Server. Only use this provider with the Connect JDBC driver embedded in WebSphere; it cannot be used with a Connect JDBC driver purchased separately from DataDirect Technologies.

   This JDBC provider supports this data source:

   `com.ibm.websphere.jdbcx.sqlserver.SQLServerDataSource.`

   Requires JDBC driver files:

sqlserver.jar
base.jar and
util.jar.

(The **spy.jar** file is optional. You need this file to enable spy logging. The **spy.jar** file for the WebSphere embedded Connect JDBC driver ships with WebSphere Application Server. All the files are located in the WAS_HOME/lib/ directory.)

Requires **DataStoreHelper** class:

com.ibm.websphere.rsadapter.WSConnectJDBCDataStoreHelper

Requires a valid authentication alias.

Requires properties:

- **serverName** The name of the server in which MS SQL Server resides. Example: *myserver.mydomain.com*
- **portNumber** The TCP/IP port that MS SQL Server uses for communication. Port 1433 is the default.
- **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

4. **WebSphere embedded ConnectJDBC driver for MS SQL Server (XA)**

WebSphere embedded ConnectJDBC 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 JDBC driver ships with WebSphere Application Server. Use this provider with the Connect JDBC driver embedded in WebSphere. Do not use it with a Connect JDBC driver purchased separately from DataDirect Technologies.

This JDBC provider supports this data source:

com.ibm.websphere.jdbcx.sqlserver.SQLServerDataSource.

Requires JDBC driver files:

sqlserver.jar
base.jar and
util.jar.

(The **spy.jar** file is optional. You need this file to enable spy logging. The **spy.jar** file for the WebSphere embedded Connect JDBC driver ships with WebSphere Application Server. All the files are located in the WAS_HOME/lib/ directory.)

Requires **DataStoreHelper** class:

com.ibm.websphere.rsadapter.WSConnectJDBCDataStoreHelper

Requires a valid authentication alias.

Requires properties:

- **serverName** The name of the server in which MS SQL Server resides. Example: *myserver.mydomain.com*
- **portNumber** The TCP/IP port that MS SQL Server uses for communication. Port 1433 is the default.
- **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

Whether you need to support one or two phase transactions with the WebSphere embedded Connect JDBC XA driver, you must install Stored Procedures for the Java Transaction API (JTA) on your machine that runs Microsoft SQL. The WebSphere Application Server installation disks provide a base level of Stored Procedures for JTA. (You can find the most current version of the software on the WebSphere Application Server-embedded DataDirect Technologies product update Web page.) Install Stored Procedures for JTA by performing the following steps:

a. Determine whether you are running the 32-bit or 64-bit MS SQL Server and select the appropriate sqljbc.dll and instjdbc.sql files.
b. Stop your MS SQL Server service.
c. Copy the sqljdbc.dll file into your %SQL_SERVER_INSTALL%\Binn\ directory.
d. Restart the MS SQL Server service.
e. Run theinstjdbc.sql script. (The script can be run by the MS SQL Server Query Analyzer or the ISQL utility).

You can download the latest patches and upgrades to the WebSphere embedded Connect JDBC driver from the following FTP site:

`ftp://ftp.software.ibm.com/software/websphere/info/tools/DataDirect/datadirect.htm`

5. **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*.

6. **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/`

7. **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*.

8. **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
```

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 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 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

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
- Styles of messaging in applications
- Using JMS interfaces to explicitly poll for messages
- Using message-driven beans to automatically retrieve messages
- ″Components of JMS support″ in the information center.
- Components of message-driven bean support
- Security considerations for asynchronous messaging

JMS providers

This topic provides an overview of the support for JMS providers by WebSphere Application Server.

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.

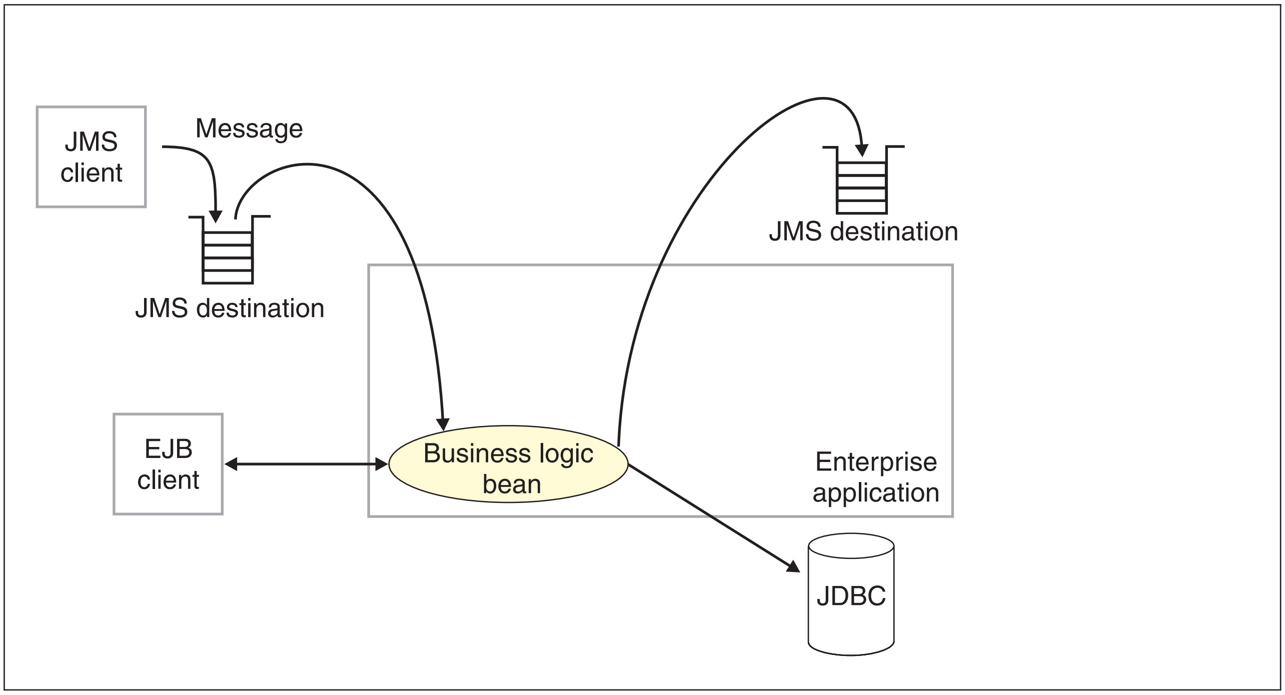


Figure 3. Asynchronous messaging using JMS. This figure shows an enterprise application polling a JMS destination to retrieve an incoming message, which it processes with a business logic bean. The business logic bean uses standard JMS calls to process the message; for example, to extract data or to 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 EJB component security.

For JMS messaging, message-driven beans can use a JMS provider that has a JCA 1.5 resource adapter, such as the default messaging provider that is part of WebSphere Application Server version 6. With a JCA 1.5 resource adapter, you deploy EJB 2.1 message-driven beans as JCA 1.5-compliant resources, to use a J2C activation specification. If the JMS provider does not have a JCA 1.5 resource adapter, such as the V5 Default Messaging and WebSphere MQ, you must configure JMS message-driven beans against a listener port (as in WebSphere Application Server version 5).

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 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 endpoints must specify the resource adapter that sends messages to the endpoint.

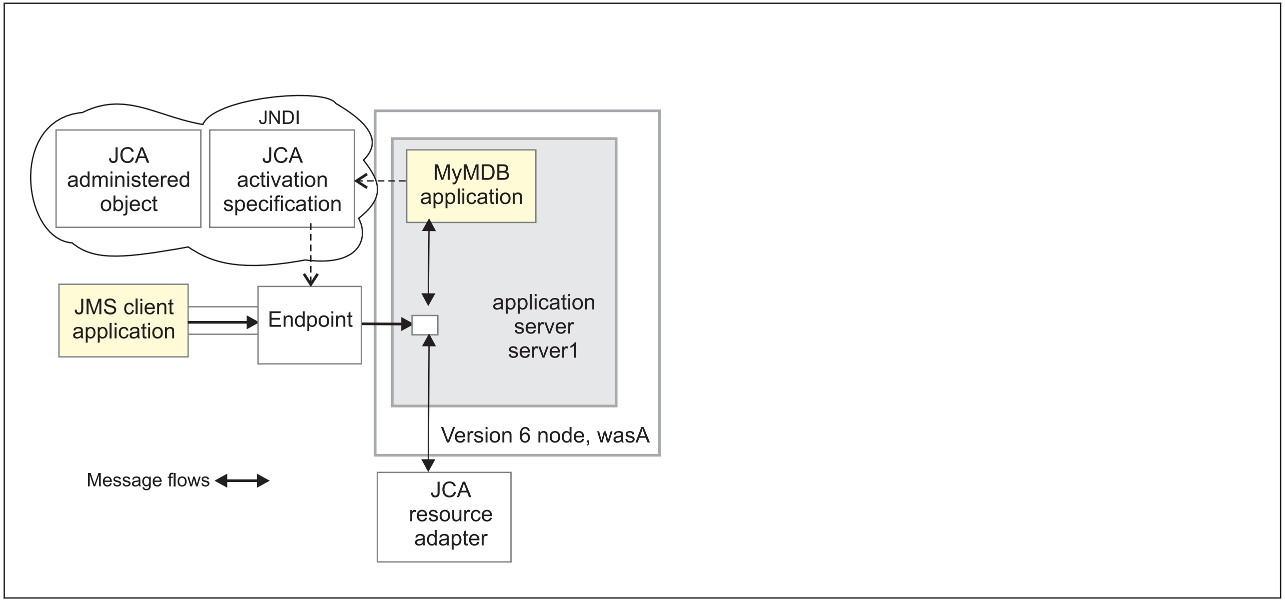


Figure 4. Message-driven bean components for a JCA resource adapter. This figure shows the main components of WebSphere support for message-driven beans for use with an external JCA resource adapter.

The administrator creates a J2C activation specification 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.

Components for 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 default messaging provider that is part of WebSphere Application Server and configure message-driven beans to use a JCA activation specification.

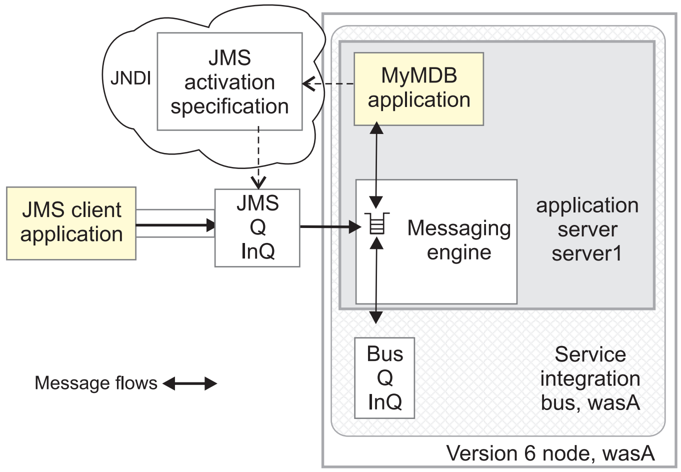


Figure 5. Message-driven bean components for the default messaging provider. This figure shows the main components of WebSphere support for message-driven beans for use with the default messaging provider.

With the default messaging provider, a message-driven bean acts as a listener on a specific JMS destination.

The administrator creates a JMS activation specification to provide information to the deployer about the configuration properties of a message-driven bean related to the processing of the inbound messages. 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 read from destinations (or endpoints) within the scope of a transaction. 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.

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.

Messaging: Resources for learning

- Sun’s Java Message Service (JMS) specification documentation.

  Provides details about the Java Message Service (JMS).
- Sun’s J2EE Connector Architecture specification (http://java.sun.com/j2ee/connector/download.html).

  Provides details about inbound messaging processing using the J2EE Connector architecture.
- J2EE specification

  Provides details about the J2EE specification, including messaging considerations.
- WebSphere MQ Using Java.

  Provides information about using JMS with WebSphere MQ as a messaging provider.
- http://www-3.ibm.com/software/ts/mqseries/library/manualsa/manuals/platspecific.html

  Provides WebSphere MQ messaging platform-specific books.
- WebSphere MQ Event Broker Web site at http://www-4.ibm.com/software/ts/mqseries/platforms/#eventb

  Provides books about WebSphere MQ Event Broker as a publish/subscribe messaging broker.
- WebSphere MQ Integrator Web site at http://www-4.ibm.com/software/ts/mqseries/platforms/#integrator

  Provides books about WebSphere MQ Integrator as a publish/subscribe messaging broker.
- IBM Publications Center

  This Web site provides a wide range of IBM publications, including publications about messaging products.

Installing and configuring a JMS provider

This topic describes the different ways that you can use JMS providers with WebSphere Application Server. A JMS provider enables use of the Java Message Service (JMS) and other message resources in WebSphere Application Server.

IBM WebSphere Application Server supports asynchronous messaging through the use of a JMS provider and its related messaging system. JMS providers must conform to the JMS specification version 1.1. To use message-driven beans the JMS provider must support the optional Application Server Facility (ASF) function defined within that specification, or support an inbound resource adapter as defined in the JCA specification version 1.5.

The service integration technologies of IBM WebSphere Application Server can act as a messaging system when you have configured a service integration bus that is accessed through the default messaging provider. This support is installed as part of WebSphere Application Server, administered through the administrative console, and is fully integrated with the WebSphere Application Server runtime.

WebSphere Application Server also includes support for the following JMS providers:

WebSphere MQ
Provided for use with supported versions of WebSphere MQ.

Generic
Provided for use with any 3rd party messaging system. If you want to use message-driven beans, the messaging system must support ASF.

For more information about the support for JMS providers, see “JMS providers” on page 1242.

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. Define 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″ in the information center.

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”
- “Using JMS resources of WebSphere MQ” on page 1286
- “Using JMS resources of a generic provider” on page 1353
- “Administering support for message-driven beans” on page 1361
- Programming to use asynchronous messaging
- Troubleshooting WebSphere messaging

Maintaining Version 5 default messaging resources

This topic is the entry-point into a set of topics about maintaining messaging resources provided for WebSphere Application Server version 5 applications by the default messaging provider.

WebSphere application server version 5 applications can use JMS resources provided by the default messaging provider of WebSphere application server version 6. You can use the WebSphere administrative console to manage the JMS connection factories and destinations for WebSphere Application Server version 5 applications. Such JMS resources are maintained as V5 Default Messaging resources.

V5 Default Messaging provides a JMS transport to a messaging engine of a service integration bus that supports the default messaging provider of WebSphere Application Server version 6. The messaging engine emulates the service of a version 5 JMS server.

You can also use the administrative console to manage a JMS server on a Version 5 node.

For more information about maintaining Version 5 default messaging resources, see the following topics:

- “Listing version 5 default messaging resources” on page 1254

- “Configuring Version 5 default JMS resources” on page 1277
- “Managing Version 5 JMS servers in a deployment manager cell” on page 1280
- “Configuring authorization security for a Version 5 default messaging provider” on page 1281

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.

Note: JMS servers make use of WebSphere MQ properties and, in general, the default values of those properties are adequate. However, if you are running high messaging loads, you may need to change some WebSphere MQ properties; for example, properties for log file locations, file pages, and buffer pages. For more information about configuring WebSphere MQ properties, see the *WebSphere MQ System Administration* book, SC33-1873, which is available from the IBM Publications Center or from the WebSphere MQ collection kit, SK2T-0730.

Name:

The name by which the JMS server is known for administrative purposes within IBM WebSphere Application Server.

This name should not be changed.

Data type	String
Units	Not applicable
Default	WebSphere Internal JMS Server
Range	Not applicable

Description:

A description of the JMS server, for administrative purposes within IBM WebSphere Application Server.

This string should not be changed.

Data type	String
Default	WebSphere Internal JMS Server

Number of threads:

The number of concurrent threads to be used by the publish/subscribe matching engine

The number of concurrent threads should only be set to a small number.

Data type	Integer
Units	Threads
Default	1
Range	Greater than or equal to 1.

Queue Names:

The names of the queues hosted by this JMS server. Each queue name must be added on a separate line.

Each queue listed in this field must have a separate queue administrative object with the same administrative name. To make a queue available to applications, define a WebSphere queue and add its name to this field on the JMS Server panel for the host on which you want the queue to be hosted.

Data type	String
Units	Queue name
Range	Each entry in this field is a queue name of up to 45 characters, which must match exactly (including use of upper- and lowercase characters) the WebSphere queue administrative object defined for the queue.

Initial State:

The state that you want the JMS server to have when it is next restarted.

Data type	Enum
Default	Started
Range	**Started** The JMS server is started automatically. **Stopped** The JMS server is not started automatically. If any deployed enterprise applications are to use JMS server functions provided by the JMS server, the system administrator must start the JMS server manually or select the Started value of this property then restart the JMS server. To restart a JMS server on a version 5 node, stop then restart that JMS server.

Version 5 WebSphere Queue connection factory collection:

Use this panel to list JMS queue connection factories for point-to-point messaging, for use by WebSphere Application Server version 5 applications. These configuration properties control how connections are created between the JMS provider and the default messaging system that it uses.

This panel shows a list of the JMS queue connection factories with a summary of their configuration properties.

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

1. In the navigation pane, click **Resources** → **JMS Providers** → **V5 Default Messaging**.
2. **(Optional)** In the content pane, change the **Scope** setting to the level at which JMS resources are visible to applications. If you define a Version 5 JMS resource at the Cell scope level, all users in the cell can look up and use that JMS resource. However, the JMS resource has only the subset of properties that apply to WebSphere Application Server Version 5. If you want to define a JMS resource at Cell level for use on non-Version 5 nodes, you should define the JMS resource for the Version 6 default messaging provider.
3. Under Additional Resources, click **WebSphere queue connection factories**. This displays a list of any existing JMS queue connection factories.

To define a new JMS queue connection factory, click **New**.

To browse or change the properties of a connection factory, select its name in the list displayed.

To act on one or more of the connection factories listed, click the check box next to the name of the connection factory, then use the buttons provided.

Version 5 WebSphere queue connection factory settings:

Use this panel to browse or change the configuration properties of the selected JMS queue connection factory for point-to-point messaging for use by WebSphere Application Server version 5 applications.

A WebSphere queue connection factory is used to create JMS connections to the default messaging provider for use by WebSphere Application Server version 5 applications.

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

1. In the navigation pane, click **Resources** → **JMS Providers** → **V5 Default Messaging**.
2. **(Optional)** In the content pane, change the **Scope** setting to the level at which JMS resources are visible to applications. If you define a Version 5 JMS resource at the Cell scope level, all users in the cell can look up and use that JMS resource.

3. Under Additional Resources, click **WebSphere queue connection factories**. This displays a list of any existing JMS queue connection factories.
4. Click the name of the JMS queue connection factory that you want to work with.

A queue connection factory for the embedded WebSphere JMS provider has the following properties:

Scope:

Specifies the level to which this resource definition is visible to applications.

Resources such as messaging providers, namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

The scope displayed is for information only, and cannot be changed on this panel. If you want to browse or change this resource (or other resources) at a different scope, change the scope on the messaging provider settings panel, then click **Apply**, before clicking the link for the type of resource.

Data type	String

Name:

The name by which this JMS queue connection factory is known for administrative purposes within IBM WebSphere Application Server. The name must be unique within the JMS connection factories across the WebSphere administrative domain.

Data type	String
Default	Null

JNDI name:

The JNDI name that is used to bind the JMS connection factory into the name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type	String

Description:

A description of this connection factory for administrative purposes within IBM WebSphere Application Server.

Data type	String
Default	Null

Category:

A category used to classify or group this connection factory, for your IBM WebSphere Application Server administrative records.

Data type	String

Node:

The WebSphere node name of the administrative node where the JMS server runs for this connection factory. Connections created by this factory connect to that JMS server.

Data type	Enum
Default	Null
Range	Pull-down list of Version 5 nodes in the WebSphere administrative domain.

Component-managed Authentication Alias:

This alias specifies a user ID and password to be used to authenticate connection to the messaging provider for application-managed authentication.

This property provides a list of the J2C authentication data entry aliases that have been defined to WebSphere Application Server. You can select a data entry alias to be used to authenticate the creation of a new connection to the JMS provider.

If you have enabled 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.

You can use the Tivoli Performance Viewer to find the optimal number of connections in a pool. If the number of concurrent waiters is greater than 0, but the CPU load is not close to 100%, consider increasing the connection pool size. If the Percent Used value is consistently low under normal workload, consider decreasing the number of connections in the pool.

Data type	Integer
Default	10
Range	0 to max int

Min Connections:

Specifies the minimum number of physical connections to maintain.

Until this number is reached, the pool maintenance thread does not discard physical connections. However, no attempt is made to bring the number of connections up to this number. If you set a value for Aged Timeout, the minimum is not maintained. All connections with an expired age are discarded.

For example if the **Min Connections** value is set to 3, and one physical connection is created, the Unused Timeout thread does not discard that connection. By the same token, the thread does not automatically create two additional physical connections to reach the **Min Connections** setting.

Data type	Integer
Default	1
Range	0 to max int

Reap Time:

Specifies the interval, in seconds, between runs of the pool maintenance thread.

For example, if **Reap Time** is set to 60, the pool maintenance thread runs every 60 seconds. The Reap Time interval affects the accuracy of the **Unused Timeout** and **Aged Timeout** settings. The smaller the interval, the greater the accuracy. If the pool maintenance thread is enabled, set the Reap Time value less than the values of Unused Timeout and Aged Timeout. When the pool maintenance thread runs, it discards any connections remaining unused for longer than the time value specified in Unused Timeout, until it reaches the number of connections specified in **Min Connections**. The pool maintenance thread also discards any connections that remain active longer than the time value specified in Aged Timeout.

The Reap Time interval also affects performance. Smaller intervals mean that the pool maintenance thread runs more often and degrades performance.

To disable the pool maintenance thread set Reap Time to 0, or set both Unused Timeout and Aged Timeout to 0. The recommended way to disable the pool maintenance thread is to set Reap Time to 0, in which case Unused Timeout and Aged Timeout are ignored. However, if Unused Timeout and Aged Timeout are set to 0, the pool maintenance thread runs, but only physical connections which timeout due to non-zero timeout values are discarded.

Data type	Integer
Units	Seconds
Default	180
Range	0 to max int

Unused Timeout:

Specifies the interval in seconds after which an unused or idle connection is discarded.

Set the Unused Timeout value higher than the Reap Timeout value for optimal performance. Unused physical connections are only discarded if the current number of connections not in use exceeds the **Min Connections** setting. For example, if the unused timeout value is set to 120, and the pool maintenance thread is enabled (Reap Time is not 0), any physical connection that remains unused for two minutes is discarded. Note that accuracy of this timeout, as well as performance, is affected by the **Reap Time** value. For more information, see Reap Time.

Data type	Integer
Units	Seconds
Default	1800
Range	0 to max int

Aged Timeout:

Specifies the interval in seconds before a physical connection is discarded.

Setting **Aged Timeout** to 0 supports active physical connections remaining in the pool indefinitely. Set the Aged Timeout value higher than the **Reap Timeout** value for optimal performance. For example, if the Aged Timeout value is set to 1200, and the Reap Time value is not 0, any physical connection that remains in existence for 1200 seconds (20 minutes) is discarded from the pool. Note that accuracy of this timeout, as well as performance, are affected by the Reap Time value. For more information, see Reap Time.

Data type	Integer
Units	Seconds
Default	0
Range	0 to max int

Purge Policy:

Specifies how to purge connections when a *stale connection* or *fatal connection error* is detected.

Valid values are **EntirePool** and **FailingConnectionOnly**. JCA data sources can have either option. WebSphere Version 4.0 data sources always have a purge policy of **EntirePool**.

Data type
String

Default
FailingConnectionOnly

Range

EntirePool
All connections in the pool are marked stale. Any connection not in use is immediately closed. A connection in use is closed and throws a *StaleConnectionException* during the next operation on that connection. Subsequent *getConnection* requests from the application result in new connections to the database opening. When using this purge policy, there is a slight possibility that some connections in the pool are closed unnecessarily when they are not stale. However, this is a rare occurrence. In most cases, a purge policy of EntirePool is the best choice.

FailingConnectionOnly
Only the connection that caused the *StaleConnectionException* is closed. Although this setting eliminates the possibility that valid connections are closed unnecessarily, it makes recovery from an application perspective more complicated. Because only the currently failing connection is closed, there is a good possibility that the next *getConnection* request from the application can return a connection from the pool that is also stale, resulting in more stale connection exceptions.

Version 5 WebSphere topic connection factory collection:

Use this panel to list JMS topic connection factories for publish/subscribe messaging, for use by WebSphere Application Server version 5 applications. These configuration properties control how connections are created between the JMS provider and the default messaging system that it uses.

This panel shows a list of JMS topic connection factories with a summary of their configuration properties.

To view this page, use the administrative console to complete the following steps:
1. In the navigation pane, click **Resources → JMS Providers → V5 Default Messaging**.
2. **(Optional)** In the content pane, change the **Scope** setting to the level at which JMS resources are visible to applications. If you define a Version 5 JMS resource at the Cell scope level, all users in the cell can look up and use that JMS resource. However, the JMS resource has only the subset of properties that apply to WebSphere Application Server Version 5. If you want to define a JMS resource at Cell level for use on non-Version 5 nodes, you should define the JMS resource for the Version 6 default messaging provider.
3. Under Additional Resources, click **WebSphere topic connection factories**. This displays a list of any existing JMS topic connection factories.

To define a new JMS topic connection factory, click **New**.

To view or change the properties of a connection factory, select its name in the list displayed.

To act on one or more of the connection factories listed, click the check box next to the name of the connection factory, then use the buttons provided.

WebSphere topic connection factory settings:

Use this panel to browse or change the configuration properties of the selected JMS topic connection factory for publish/subscribe messaging by WebSphere Application Server version 5 applications.

A WebSphere topic connection factory is used to create JMS connections to the default messaging provider for use by WebSphere Application Server version 5 applications.

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

1. In the navigation pane, click **Resources** → **JMS Providers** → **V5 Default Messaging**.
2. **(Optional)** In the content pane, change the **Scope** setting to the level at which JMS resources are visible to applications. If you define a Version 5 JMS resource at the Cell scope level, all users in the cell can look up and use that JMS resource.
3. Under Additional Resources, click **WebSphere topic connection factories**. This displays a list of any existing JMS topic connection factories.
4. Click the name of the JMS topic connection factory that you want to work with.

A JMS topic connection factory for use with the Version 5 default messaging provider has the following properties.

Scope:

Specifies the level to which this resource definition is visible to applications.

Resources such as messaging providers, namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

The scope displayed is for information only, and cannot be changed on this panel. If you want to browse or change this resource (or other resources) at a different scope, change the scope on the messaging provider settings panel, then click **Apply**, before clicking the link for the type of resource.

Data type	String

Name:

The name by which this JMS topic connection factory is known for administrative purposes within IBM WebSphere Application Server. The name must be unique within the JMS connection factories across the WebSphere administrative domain.

Data type	String
Default	Null

JNDI name:

The JNDI name that is used to bind the topic connection factory into the name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type	String

Description:

A description of this topic connection factory for administrative purposes within IBM WebSphere Application Server.

Data type	String
Default	Null

Category:

A category used to classify or group this topic connection factory, for your IBM WebSphere Application Server administrative records.

Data type	String

Node:

The WebSphere node name of the administrative node where the JMS server runs for this connection factory. Connections created by this factory connect to that JMS server.

Data type	Enum
Default	Null
Range	Pull-down list of nodes in the WebSphere administrative domain.

Port:

Which of the two ports that connections use to connect to the JMS server. The QUEUED port is for full-function JMS publish/subscribe support, the DIRECT port is for non-persistent, non-transactional, non-durable subscriptions only.

Note: Message-driven beans cannot use the direct listener port for publish/subscribe support. Therefore, any topic connection factory configured with **Port** set to `Direct` cannot be used with message-driven beans.

Data type	Enum
Units	Not applicable
Default	QUEUED
Range	**QUEUED** The listener port used for full-function JMS-compliant, publish/subscribe support. **DIRECT** The listener port used for direct TCP/IP connection (non-transactional, non-persistent, and non-durable subscriptions only) for publish/subscribe support.

Component-managed Authentication Alias:

This alias specifies a user ID and password to be used to authenticate connection to the messaging provider for application-managed authentication.

This property provides a list of the J2C authentication data entry aliases that have been defined to WebSphere Application Server. You can select a data entry alias to be used to authenticate the creation of a new connection to the JMS provider.

If you have enabled 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.

- Configuring a Version 5 default JMS queue connection factory
- Configuring a Version 5 default JMS topic connection factory
- Configuring a Version 5 default JMS queue destination
- Configuring a Version 5 default JMS topic destination

Configuring a Version 5 queue connection factory:

Use this task to browse or change the properties of a JMS queue connection factory for point-to-point messaging with the default messaging provider on a Version 5 node in the deployment manager cell. This task contains an optional step for you to create a new JMS queue connection factory.

To configure a JMS queue connection factory for use by WebSphere application server version 5 applications, use the administrative console to complete the following steps:

1. Display the version 5 default messaging provider. In the navigation pane, expand **Resources** → **JMS 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 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 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 `*WAS_install_root*\config\cells\*your_cell_name*\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. Please note the file is in Unicode, which requires a binary FTP to the host from a workstation.
2. Edit the queue-admin-userids element to create a list of userids with administrative access to all queues. Administrative access is needed to create queues and perform other administrative activities on queues. For example, consider the following queue-admin-userids section:

   ```
   <queue-admin-userids>
       <userid>adminid1</userid>
       <userid>adminid2</userid>
   </queue-admin-userids>
   ```

   In this example the userids adminid1 and adminid2 are defined to have administrative access to all queues.

3. Edit the queue-default-permissions element to define the default queue access permissions. These permissions are used for queues for which you do not define specific permissions (in queue sections). If this section is not specified, then access permissions exist only for those queues for which you have specifically created queue elements.

   For example, consider the following queue-default-permissions element:

   ```
   <queue-default-permissions>
     <permission>write</permission>
   </queue-default-permissions>
   ```

   In this example the default access permission for all queues is **write**. This can be overridden for a specific queue by creating a queue element that sets its access permission to **read**.

4. If you want to define specific access permissions for a queue, create a queue element, then define the following elements:

   For example, consider the following queue element:

   ```
   <queue>
     <name>q1</name>
     <public>
     </public>
     <authorize>
       <userid>useridr</userid>
       <permission>read</permission>
     </authorize>
     <authorize>
       <userid>useridw</userid>
       <permission>write</permission>
     </authorize>
     <authorize>
       <userid>useridrw</userid>
       <permission>read</permission>
       <permission>write</permission>
     </authorize>
   </queue>
   ```

   In this example for the queue q1, the userid useridr has read permission, the userid useridw has write permission, the userid useridrw has both read and write permissions, and all other userids have no access permissions (<public></public>).

5. Edit topic elements to define the access permissions for publish/subscribe topic destinations.

   For topics, you can grant and deny access permissions. Full permission inheritance is supported on topics. If you do not define specific access permissions for a userid on a specific topic then permissions are inherited first from the public permissions on that topic then from the parent topic. The inheritance of access permissions continues until the root topic from which the root permissions are assumed.

   a. If you want to define default access permissions for the root topic, edit a topic element with an empty name element. If you omit such a topic section, topics have no default topic permissions other than those defined by specific topic elements. For example, consider the following topic element for the root topic:

      ```
      <topic>
         <name></name>
         <public>
           <permission>+pub</permission>
         </public>
       </topic>
      ```

      In this example, the default access permission for all topics is set to publish. This can be overridden by other topic elements for specific topic names.

   b. If you want to define access permissions for a specific topic, create a topic element with the name for the topic then define the access permissions in the public and authorize elements of the topic element. For example, consider the following topic section:

```
<topic>
  <name>a/b/c</name>
  <public>
    <permission>+sub</permission>
  </public>
  <authorize>
    <userid>useridpub</userid>
    <permission>+pub</permission>
  </authorize>
</topic>
```

In this example, the subscribe permission is granted to anyone accessing any topic whose name starts with a/b/c. Also, the userid useridpub is granted publish permission for any topic whose name starts with a/b/c.

6. Save the integral-jms-authorizations.xml file.

If the dynamic update setting is selected, changes to the integral-jms-authorizations.xml file become active when the changed file is saved, so there is no need to stop and restarted the JMS server. If the dynamic update setting is not selected, you need to stop and restart the JMS server to make changes active.

Authorization settings for Version 5 default JMS resources:

Use the integral-jms-authorisations.xml file to view or change the authorization settings for JMS resources owned by the default messaging provider on WebSphere Application Server version 5 nodes.

Authorization to access default JMS resources owned by the default messaging provider on WebSphere Application Server nodes is controlled by the following settings in the *was_install*\config\cells\your_cell_name\integral-jms-authorisations.xml file.

This structure of the settings in integral-jms-authorisations.xml is shown in the following example. Descriptions of these settings are provided after the example. To configure authorization settings, follow the instructions provided in Configuring authorization security for the Version 5 JMS providers

```
<integral-jms-authorizations>

  <dynamic-update>true</dynamic-update>

  <queue-admin-userids>
    <userid>adminid1</userid>
    <userid>adminid2</userid>
  </queue-admin-userids>

  <queue-default-permissions>
    <permission>write</permission>
  </queue-default-permissions>

  <queue>
    <name>q1</name>
    <public>
    </public>
    <authorize>
      <userid>useridr</userid>
      <permission>read</permission>
    </authorize>
    <authorize>
      <userid>useridw</userid>
      <permission>write</permission>
    </authorize>
  </queue>

  <queue>
    <name>q2</name>
    <public>
      <permission>write</permission>
    </public>
    <authorize>
      <userid>useridr</userid>
      <permission>read</permission>
    </authorize>
  </queue>

  <topic>
    <name></name>
    <public>
      <permission>+pub</permission>
    </public>
  </topic>

  <topic>
    <name>a/b/c</name>
    <public>
      <permission>+sub</permission>
    </public>
    <authorize>
      <userid>useridpub</userid>
      <permission>+pub</permission>
    </authorize>
  </topic>

</integral-jms-authorizations>
```

dynamic-update: Controls whether or not the JMS Server checks dynamically for updates to this file.

true (Default) Enables dynamic update support.

false Disables dynamic update checking and improves authorization performance.

queue-admin-userids: This element lists those userids with administrative access to all Version 5 default queue destinations. Administrative access is needed to create queues and perform other administrative activities on queues. You define each userid within a separate userid sub element:

<userid>*adminid***</userid>**

Where *adminid* is a user ID that can be authenticated by IBM WebSphere Application Server.

queue-default-permissions: This element defines the default queue access permissions that are assumed if no permissions are specified for a specific queue name. These permissions are used for queues for which you do not define specific permissions (in queue elements). If this element is not specified, then no access permissions exist unless explicitly authorized for individual queues.

You define the default permission within a separate permission sub element:

<permission>*read-write***</permission>**

Where *read-write* is one of the following keywords:

read By default, userids have read access to Version 5 default queue destinations.

write By default, userids have write access to Version 5 default queue destinations.

queue: This element contains the following authorization settings for a single queue destination:

name The name of the queue.

public The default public access permissions for the queue. This is used only for those userids that have no specific authorize element. If you leave this element empty, or do not define it at all, only those userids with authorize elements can access the queue.

You define each default permission within a separate permission element.

authorize

The access permissions for a specific userid. Within each authorize element, you define the following elements:

userid The userid that you want to assign a specific access permission.

permission

An access permission for the associated userid.

You define each permission within a separate permission element. Each permission element can contain the keyword read or write to define the access permission.

For example, consider the following queue element:

```
<queue>
  <name>q1</name>
  <public>
  </public>
  <authorize>
    <userid>useridr</userid>
    <permission>read</permission>
  </authorize>
  <authorize>
    <userid>useridw</userid>
    <permission>write</permission>
  </authorize>
  <authorize>
    <userid>useridrw</userid>
    <permission>read</permission>
    <permission>write</permission>
  </authorize>
</queue>
```

topic: This element contains the following authorization settings for a single topic destination:

Each topic element has the following sub elements:

name The name of the topic, without wildcards or other substitution characters.

public The default public access permissions for the topic. This is used only for those userids that have no specific authorize element. If you leave this element empty, or do not define it at all, only those userids with authorize elements can access the topic.

You define each default permission within a separate permission element.

authorize

The access permissions for a specific userid. Within each authorize element, you define the following elements:

userid The userid that you want to assign a specific access permission.

permission

An access permission for the associated userid.

You define each permission within a separate permission element. Each permission element can contain one of the following keywords to define the access permission:

+pub Grant publish permission

+sub Grant subscribe permission

+persist

Grant persist permission

-pub Deny publish permission

-sub Deny subscribe permission

-persist

Deny persist permission

JMS components on version 5 nodes

To provide messaging support on a WebSphere Application Server version 5 node, there is at most one JMS server and some number of JMS resources configured for the default messaging JMS provider on that node.

A JMS server on a version 5 node serves the JMS resources (connection factories and destinations) for that node. The JMS server is managed as a separate process to application servers on the same node. Any application server within the domain can access JMS resources served by any JMS server on any node in the domain.

A connection factory encapsulates the configuration properties used to create connections with the JMS provider, to enable applications to access JMS destinations.

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”
- “Listing JMS resources for WebSphere MQ” on page 1288
- “Configuring JMS resources for the WebSphere MQ messaging provider” on page 1346

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.

If you have a messaging infrastructure based on WebSphere MQ, 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.

- **(UNIX platforms only)** Before you install WebSphere MQ on a UNIX platform, create and mount a journalized file system called /var/mqm for your messaging working data. Use a partition strategy with a separate volume for the WebSphere MQ data. This means that other system activity is not affected if a large amount of messaging work builds up in /var/mqm. You can also create separate file systems for your log data (var/mqm/log) and error files (var/mqm/errors). You should store log files on a different physical volume from the messaging queues (var/mqm). This ensures data integrity in the case of a hardware failure. If you are creating separate file systems, allow a minimum of 30 MB of storage for /var/mqm, 20 MB of storage for /var/mqm/log, and 4 MB of storage for /var/mqm/errors.
  - The /var file system is used to store all the security logging information for the system, and is used to store the temporary files for email and printing. Therefore, it is critical that you maintain free space in /var for these operations. If you do not create a separate file system for messaging data, and /var fills up, all security logging will be stopped on the system until some free space is available in /var. Also, email and printing will no longer be possible until some free space is available in /var.
  - It is not recommended to install Rational Application Developer and WebSphere Application Server on the same machine when using WebSphere MQ.
  - For more information a security logging issue, see ″Security and WebSphere MQ″ in the information center.
  - For more information about creating file systems for WebSphere MQ, and WebSphere MQ space requirements for /var file systems, see the section “Preparing for Installation: Creating WebSphere MQ file systems” in the appropriate WebSphere MQ *Quick Beginnings* book.
  - For other installation prerequisites, see the appropriate WebSphere MQ *Quick Beginnings* book, as follows:

- *WebSphere MQ for Windows, Quick Beginnings*, GC34-6073
- *WebSphere MQ for AIX, Quick Beginnings*, GC34-6076
- *WebSphere MQ for Solaris, Quick Beginnings*, GC34-6075
- *WebSphere MQ for HP-UX, Quick Beginnings*, GC34-6077
- *WebSphere MQ for Linux for Intel and Linux for zSeries, Quick Beginnings*, GC34-6078

You can get these books from the WebSphere MQ messaging platform-specific books Web page at http://www-3.ibm.com/software/ts/mqseries/library/manualsa/manuals/platspecific.html

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
2. If you want to use WebSphere MQ - Publish/Subscribe support, you need to provide a Publish/Subscribe broker.

   For example, you can do this by using either WebSphere MQ Event Broker or WebSphere MQ Integrator (formerly MQSeries Integrator). For more information about these products, see the following Web sites:
   - WebSphere MQ Event Broker Web site at http://www-4.ibm.com/software/ts/mqseries/platforms/#eventb
   - WebSphere MQ Integrator Web site at http://www-4.ibm.com/software/ts/mqseries/platforms/#integrator
3. Follow the WebSphere MQ instructions for verifying your installation setup.
4. AIX For AIX, see the WebSphere MQ readme.txt for additional steps.
5. If you want to install IBM WebSphere Application Server on the same host as WebSphere MQ, and have not yet done so, install IBM WebSphere Application Server.
6. 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.

(UNIX platforms only) Restrict access to the messaging errors directories and logging files, by using the following commands. This is part of the procedure to secure the directories and log files needed for WebSphere MQ, as described in Securing messaging directories and log files.

1. For the /var/mqm/errors directory:

```
chmod 3777 /var/mqm/errors
chown mqm:mqm /var/mqm/errors

touch /var/mqm/errors/AMQERR01.LOG
chown mqm:mqm /var/mqm/errors/AMQERR01.LOG
chmod 666 /var/mqm/errors/AMQERR01.LOG

touch /var/mqm/errors/AMQERR02.LOG
chown mqm:mqm /var/mqm/errors/AMQERR02.LOG
chmod 666 /var/mqm/errors/AMQERR02.LOG

touch /var/mqm/errors/AMQERR03.LOG
chown mqm:mqm /var/mqm/errors/AMQERR03.LOG
chmod 666 /var/mqm/errors/AMQERR03.LOG
```

2. For the /var/mqm/qmgrs/@SYSTEM/errors directory:

```
chmod 3777 /var/mqm/qmgrs/@SYSTEM/errors
chown mqm:mqm /var/mqm/qmgrs/@SYSTEM/errors

touch /var/mqm/qmgrs/@SYSTEM/errors/AMQERR01.LOG
chown mqm:mqm /var/mqm/qmgrs/@SYSTEM/errors/AMQERR01.LOG
chmod 666 /var/mqm/qmgrs/@SYSTEM/errors/AMQERR01.LOG

touch /var/mqm/qmgrs/@SYSTEM/errors/AMQERR02.LOG
chown mqm:mqm /var/mqm/qmgrs/@SYSTEM/errors/AMQERR02.LOG
chmod 666 /var/mqm/qmgrs/@SYSTEM/errors/AMQERR02.LOG

touch /var/mqm/qmgrs/@SYSTEM/errors/AMQERR03.LOG
chown mqm:mqm /var/mqm/qmgrs/@SYSTEM/errors/AMQERR03.LOG
chmod 666 /var/mqm/qmgrs/@SYSTEM/errors/AMQERR03.LOG
```

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** 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:

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.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. 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.htm

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″ in the information center.

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 1346.

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 1346.

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 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 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.

Securing WebSphere MQ messaging directories and log files

Use this task to restrict access to the /var/mqm directories and log files needed for WebSphere MQ as a JMS provider.

You need to set the permissions described in this topic, to reduce the risk of severe security exposures.

Note: The /var file system is used to store all the security logging information for the system, and is used to store the temporary files for email and printing. Therefore, it is critical that you maintain free space in /var for these operations and prevent unauthorized access to the file system. If you do not create a separate file system for messaging data, and /var fills up, all security logging will be stopped on the system until some free space is available in /var. Also, email and printing will no longer be possible until some free space is available in /var.

This procedure involves steps that you complete at different stages of installing and using IBM WebSphere Application Server, as described below. The steps are also described at appropriate points in other tasks, but are collected here for completeness.

This procedure applies only to the ordinary UNIX file system. If your site uses access-control lists, secure the files by using that mechanism. Any site-specific requirements can affect the desired owner, group and corresponding privileges. For example, on AIX, complete the following steps:

1. Before installing WebSphere MQ, create and mount a journalized file system called /var/mqm. Use a partition strategy with a separate volume for the messaging data. This means that other system activity is not affected if a large amount of messaging work builds up in /var/mqm.

2. Install WebSphere MQ as a messaging provider.

   As part of this stage, the installation program creates the /var/mqm/errors and /var/mqm/qmgrs/@SYSTEM/errors directories used to hold messaging logging files.

3. Restrict access to the /var/mqm/errors directory and the logging files, by using the following commands:

   ```
   chmod 3777 /var/mqm/errors
   chown mqm:mqm /var/mqm/errors

   touch /var/mqm/errors/AMQERR01.LOG
   chown mqm:mqm /var/mqm/errors/AMQERR01.LOG
   chmod 666 /var/mqm/errors/AMQERR01.LOG

   touch /var/mqm/errors/AMQERR02.LOG
   chown mqm:mqm /var/mqm/errors/AMQERR02.LOG
   chmod 666 /var/mqm/errors/AMQERR02.LOG

   touch /var/mqm/errors/AMQERR03.LOG
   chown mqm:mqm /var/mqm/errors/AMQERR03.LOG
   chmod 666 /var/mqm/errors/AMQERR03.LOG
   ```

4. Restrict access to the /var/mqm/qmgrs/@SYSTEM/errors directory and the logging files, by using the following commands:

   ```
   chmod 3777 /var/mqm/qmgrs/@SYSTEM/errors
   chown mqm:mqm /var/mqm/qmgrs/@SYSTEM/errors

   touch /var/mqm/qmgrs/@SYSTEM/errors/AMQERR01.LOG
   chown mqm:mqm /var/mqm/qmgrs/@SYSTEM/errors/AMQERR01.LOG
   chmod 666 /var/mqm/qmgrs/@SYSTEM/errors/AMQERR01.LOG

   touch /var/mqm/qmgrs/@SYSTEM/errors/AMQERR02.LOG
   chown mqm:mqm /var/mqm/qmgrs/@SYSTEM/errors/AMQERR02.LOG
   chmod 666 /var/mqm/qmgrs/@SYSTEM/errors/AMQERR02.LOG

   touch /var/mqm/qmgrs/@SYSTEM/errors/AMQERR03.LOG
   chown mqm:mqm /var/mqm/qmgrs/@SYSTEM/errors/AMQERR03.LOG
   chmod 666 /var/mqm/qmgrs/@SYSTEM/errors/AMQERR03.LOG
   ```

5. For each application server that uses JMS resources provided by WebSphere MQ, restrict access to the server's /var/mqm/qmgrs/*long_server_name*/errors directory and its messaging logging files. You should restrict access to the server's directory and logging files, as soon after creating the application server as possible.

   To restrict access to the server's directory and logging files, use the following commands:

   ```
   chmod 3775 /var/mqm/qmgrs/long_server_name/errors
   chown mqm:mqm /var/mqm/qmgrs/long_server_name/errors

   touch /var/mqm/qmgrs/long_server_name/errors/AMQERR01.LOG
   chown mqm:mqm /var/mqm/qmgrs/long_server_name/errors/AMQERR01.LOG
   chmod 666 /var/mqm/qmgrs/long_server_name/errors/AMQERR01.LOG

   touch /var/mqm/qmgrs/long_server_name/errors/AMQERR02.LOG
   chown mqm:mqm /var/mqm/qmgrs/long_server_name/errors/AMQERR02.LOG
   chmod 666 /var/mqm/qmgrs/long_server_name/errors/AMQERR02.LOG

   touch /var/mqm/qmgrs/long_server_name/errors/AMQERR03.LOG
   chown mqm:mqm /var/mqm/qmgrs/long_server_name/errors/AMQERR03.LOG
   chmod 666 /var/mqm/qmgrs/long_server_name/errors/AMQERR03.LOG
   ```

   Where *long_server_name* is the long name assigned to the server, in the following form: `WAS_nodename_server_name`. For example, if you created an application server called `server1` to run on the node called `appnode1`, the long server name would be: `WAS_appnode1_server1`.

This task has restricted access to the /var/mqm directories and log files needed for WebSphere MQ as a JMS provider, such that only the user ID mqm or members of the mqm user group have write access.

Using JMS resources of a generic provider

This topic is the entry-point into a set of topics about enabling WebSphere applications to use JMS resources provided by a generic messaging provider (other than a WebSphere default messaging provider or WebSphere MQ).

You can install a messaging provider other than the default messaging provider or WebSphere MQ. WebSphere applications can use the JMS 1.1 interfaces or JMS 1.0.2 interfaces to access JMS resources provided by the generic messaging provider, in addition to JMS resources provided by the default messaging provider or WebSphere MQ (if installed).

You can use the WebSphere administrative console to administer the JMS connection factories and destinations provided by generic messaging providers.

In a mixed-version WebSphere Application Server deployment manager cell, you can administer generic messaging resources on both Version 6 and Version 5 nodes. For Version 5 nodes, the administrative console presents the subset of resources and properties that are applicable to WebSphere Application Server Version 5.

For more information about using a generic messaging provider to WebSphere Application Server, see the following topics:
- Defining a generic messaging provider
- “Displaying administrative lists of generic messaging resources” on page 1354
- “Configuring JMS resources for a generic messaging provider” on page 1360

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 1360.

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″ in the information center.

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 1364
- 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.

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 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 Activation specifications 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 activation specification, 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**.

JNDI Name:

Specifies the JNDI name of the administered object.

Data type	String

Name:

Specifies display name assigned to this 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 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, Thread Pool, and Custom Properties for the message listener service.

To view this administrative console page, click **Servers** → **Application Servers** → *application_server* → **[Communications] Messaging** → **Message Listener Service**

Custom Properties:

An optional set of name and value pairs for custom properties of the message listener service.

You can use the Custom properties page to define the following properties for use by the message listener service.

- NON.ASF.RECEIVE.TIMEOUT
- MQJMS.POOLING.TIMEOUT
- MQJMS.POOLING.THRESHOLD
- MAX.RECOVERY.RETRIES
- RECOVERY.RETRY.INTERVAL

Message listener port collection:

The message listener ports configured in the administrative domain

This panel displays a list of the message listener ports configured in the administrative domain. Each listener port is used with a message-driven bean to automatically receive messages from an associated JMS destination. You can use this panel to add new listener ports or to change the properties of existing listener ports. 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. **Note:** • 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 the listener service tries to get a message from a listener port before the associated listener is stopped, in the range 0 through 2147483647.

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 the listener service to get a message from a listener port.

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 Additional Properties, click **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 Additional Properties, click **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 EJB 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 Additional Properties, click **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 Additional Properties, click **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 described below:

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. See ″Assembling applications″ in the information center. 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. See ″Assembling applications″ in the information center. 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 1387 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 1388 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 entries collection

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 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

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.

Security

Securing applications and their environments

WebSphere Application Server supports the 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 1394. 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 1404. For example, during this step, you learn how to protect security configurations after you install the product.
3. Migrate your existing security systems. For more information, see ″Migrating security configurations from previous releases″ in the information center.
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 those included in the topic ″Assembling applications″ in the information center, 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 1767.

   Deployer link entities referred to in an enterprise application are mapped to the runtime environment. The deployer:

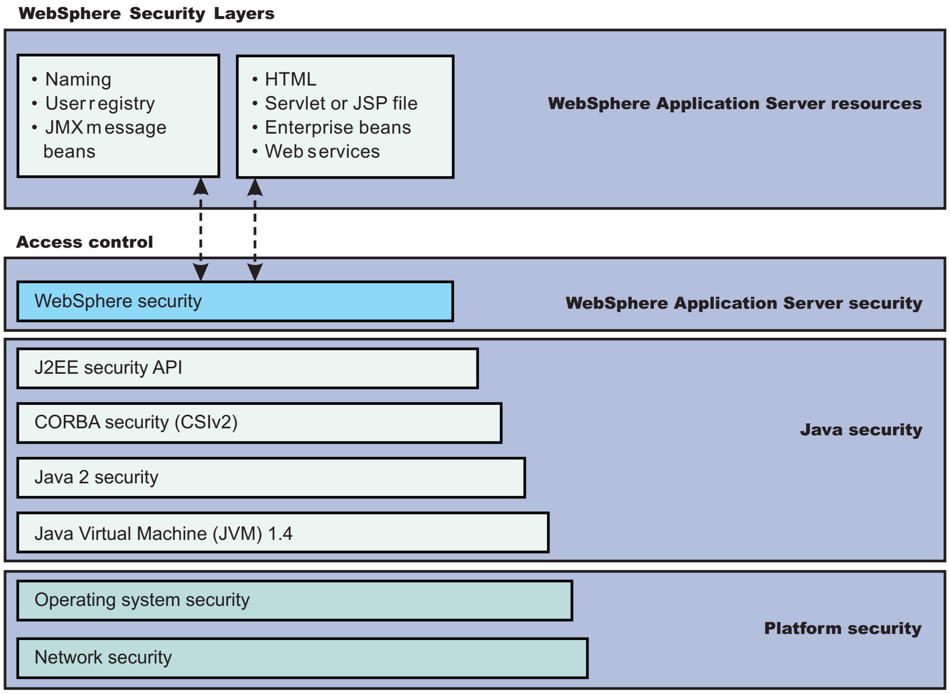
   - Maps actual users and groups to application roles
   - Installs the enterprise application into the environment
   - Makes the final adjustments needed to run the application
7. Test secured applications. For more information, see Testing security.
8. Manage security configurations. For more information, see ″Administering security″ in the information center.
9. Improve performance by tuning security configurations. For more information, see ″Tuning security configurations″ in the information center.
10. Troubleshoot security configurations. For more information, see ″Troubleshooting security configurations″ in the information center.

Your applications and production environment are secured.

See ″Security: Resources for learning″ in the information center for more information on the WebSphere Application Server security architecture.

Planning to secure your environment

There are several communication links from a browser on the Internet, 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. These services include the file system security support that secure sensitive files in the product installation for WebSphere Application Server. The system administrator can configure the product to obtain authentication information directly from the operating system user registry.
- **Network Security** - The Network Security layers provide transport level authentication and message integrity and encryption. You can configure the communication between separate application servers to use Secure Sockets Layer (SSL) and HTTPS. Additionally, you can use IP Security and Virtual Private Network (VPN) for added message protection.
- **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.
- **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.
- **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 1407.

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.

After installation, the administrative console HTTPS port is configured to use **DummyServerKeyFile.jks** and **DummyServerTrustFile.jks** with the default self-signed certificate. Using the dummy key and trust file certificate is not safe and you need to generate your own certificate to replace dummy ones immediately. It is more secure if you first enable global security and complete other configuration tasks after global security is enforced.

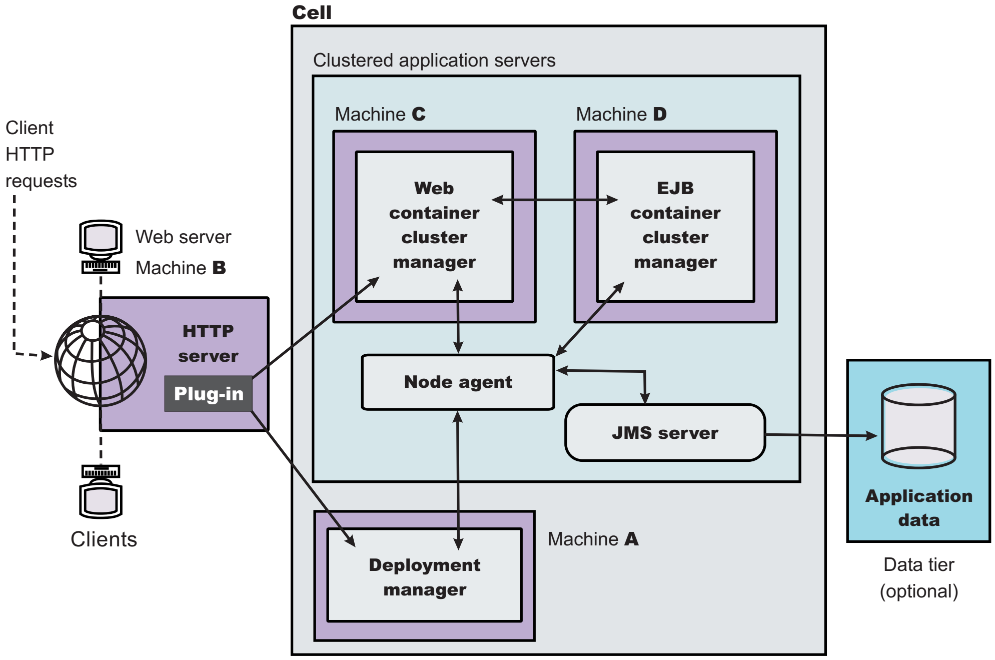


Figure 6. Multiple-tier business computing environment.

Global and administrative security:

WebSphere Application Servers interact with each other through CSIv2 and Secure Authentication Services (SAS) security protocols as well as 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 HTTP protocol and the RMI/IIOP connector uses the TCP/IP protocol. When global security is enabled, the SOAP JMX connector always uses 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.

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″ in the information center.

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 data 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 global security is enabled and application server security is not 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.

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″ and “Assigning users and groups to roles” on page 1768.

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 9. J2EE security permissions set for Web components

Security Permission	Target	Action
java.lang.RuntimePermission	loadLibrary	
java.lang.RuntimePermission	queuePrintJob	
java.net.SocketPermission	*	connect
java.io.FilePermission	*	read, write
java.util.PropertyPermission	*	read

Table 10. J2EE 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 implementation is 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).

`${USER_INSTALL_ROOT}/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_INSTALL_ROOT}/profiles/*profile_name***/config/cells/***cell_name***/nodes/***node_name***/spi.policy**
Used for embedded resources defined in the resources.xml file, such as the Java Message Service (JMS), JavaMail, and JDBC drivers.

${WAS_INSTALL_ROOT}/profiles/*profile_name***/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_INSTALL_ROOT}/profiles/*profile_name***/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.

Note: Grant extra permissions to an application after careful consideration because of the potential of compromising the system integrity.

WebSphere Application Server uses a permission filtering policy file to alert users when an application requires permissions that are on the filter list during application installation and causes the offended 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.

WebSphere Application Server Version 4 supported Java 2 Security, but enforced only three permissions checking against exitVM, create and set the security manager. Other permission checking is disabled by default.

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 quickly, you might temporarily give those applications java.security.AllPermission in the was.policy file. It is recommended to test or make those applications Java 2 Security ready; 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, refer to ″Migrating Java 2 security policy″ in the information center.

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 with

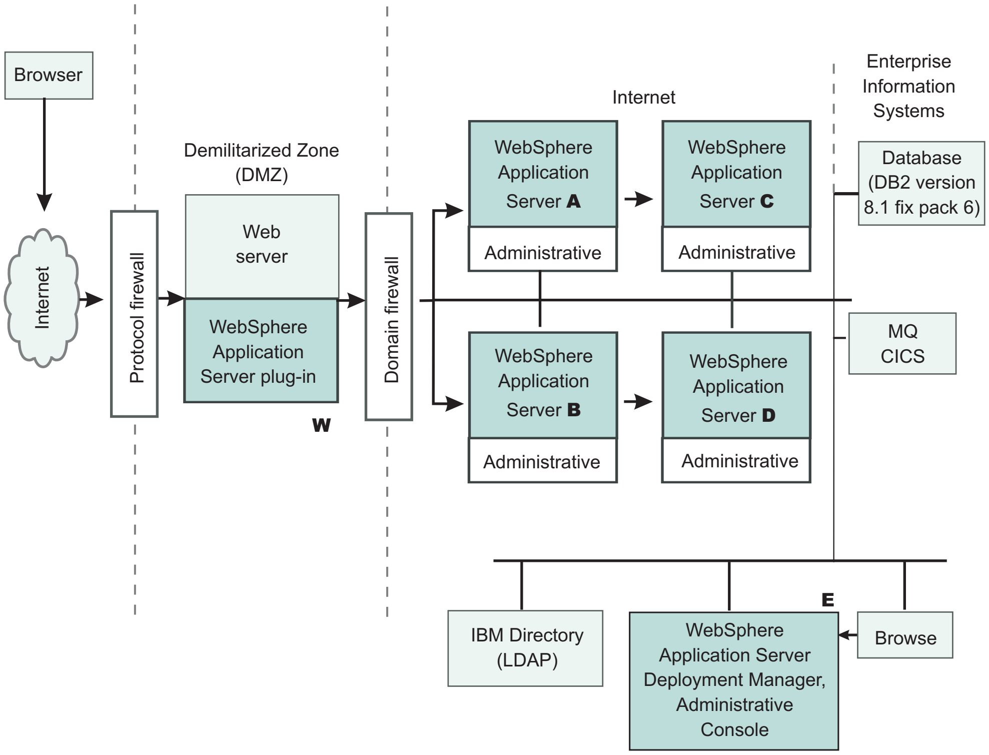
AllPermission not only have access to sensitive system resources, but also WebSphere Application Server run-time resources and can potentially cause damage to both. In cases where an application can be trusted to be safe, WebSphere Application Server allows Java 2 Security to be disabled on a per application 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.
- Because WebSphere Application Server security identity is given to the administrator role, only users with the administrator role can disable global security, to change server ID and password, and to map users and groups to administrative roles, and so on.

Other Runtime resources: Other WebSphere Application Server run time resources are protected by a similar mechanism as described previously. It is very important to enable WebSphere Application Server global security and to enforce Java 2 Security. J2EE Specification defines several authentication methods for Web components: HTTP Basic Authentication, Form-Based Authentication, and HTTPS Client Certificate Authentication. When you use client certificate login, it is more convenient for the browser client if the Web resources have integral or confidential data constraint. If a browser uses HTTP to access the Web resource, the Web container automatically redirects the browser to the HTTPS port. The CSIv2 security protocol also supports client certificate authentication. You can also use SSL client authentication to setup secure communication among a selected set of servers based on a trust relationship.

If you start from the WebSphere Application Server plug-in at the Web server, you can configure SSL mutual authentication between it and the WebSphere Application Server HTTPS server. When using a self-signed certificate, you can restrict the WebSphere Application Server plug-in to communicate with only the selected two WebSphere Application Server servers as shown in the following figure. 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 the WebSphere Application Server plug-in **W**. You can generate three self-signed certificates using the IKEYMAN and the certificate management utilities. For example, 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**. For more information on IKEYMAN, refer to ″Starting the key management utility (iKeyman)″ in the information center.


Browser
Internet
Protocol firewall
Demilitarized Zone (DMZ)
Web server
WebSphere Application Server plug-in
W
Domain firewall
Internet
WebSphere Application Server A
Administrative
WebSphere Application Server C
Administrative
WebSphere Application Server B
Administrative
WebSphere Application Server D
Administrative
Enterprise Information Systems
Database (DB2 version 8.1 fix pack 6)
MQ CICS
IBM Directory (LDAP)
E
WebSphere Application Server Deployment Manager, Administrative Console
Browse

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

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 certificate 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 file. WebSphere Application Server does not provide a user registry configuration or management utility. In addition, it does not dictate the user registry password policy. It is recommended that you use the password policy recommended by your user registry, including the password length and expiration period.

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 in the information center.
   - Authentication protocol for EJB security
     - Supported authentication protocols
     - Common Secure Interoperability Version 2 features
     - Identity assertion
   - Authentication mechanisms
     - Lightweight Third Party Authentication settings

  - Trust associations
  - Single signon (SSO)
- User registries
  - Local operating system user registries
  - Lightweight Directory Access Protocol
- Custom user registries
- Java 2 security
  - Java 2 security policy files
- Java Authentication and Authorization Service (JAAS)
  - Programmatic login
- J2EE Connector security
- Access control exception
  - Role-based authorization
  - Administrative console and naming service authorization
- Secure Socket Layer (SSL)
  - Authenticity
  - Confidentiality
  - Integrity

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 the cell, you automatically inherit both the user registry and the authentication mechanism of the 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 1768.

Another major issue 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).

See the article, ″Managing digital certificates″ in the information center.

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 the addNode command for more information on the addNode syntax.

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 can communicate with the keystore files from the Deployment Manager and vice versa. When using the default `DummyServerKeyFile` and `DummyServerTrustFile`, you should not see this problem as these are already able to communicate. However, never use these dummy files in a production environment or anytime sensitive data is being transmitted.
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 in the information center 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.

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
     ```

A sample file named `wsserver.key` also contains these instructions. After installation, you can locate this sample file in the *install_root*/`properties` directory. You can use or modify the sample file as needed for testing.

Note: You can place the login key file anywhere on a host machine running the application server. However, it is recommended that you place the login key file under a securable file system .

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 file is a key database file that contains the public keys (See “Creating login key files” on page 1403 for information about how to create a new keystore file.)

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″ in the information center.
2. Add the public key from the WebSphere Application Server as a signer certificate into the requesting WebSphere Application Server truststore file. For details, see the related information about how to import signer certificates.

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 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.
6. Specify a security property named com.ibm.CORBA.keyFileName for the absolute path of the login key file created earlier.
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. “Securing your environment before installation” on page 1405. This step describes how to install WebSphere Application Server with the proper authority.
2. Install the WebSphere Application Server. This step describes how to install WebSphere Application Server as the root user on a UNIX platform or as an administrator on a Windows platform.

   During installation you are prompted to migrate 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 before installation

The following instructions explain how to perform a product installation with proper authority on UNIX platforms, Linux platforms, Solaris operating environments, and Windows platforms.

UNIX platforms

On UNIX platforms, log on as **root** and verify that the umask value is **022**.

To verify that the umask value is **022**, execute the **umask** command.

To set up the umask value as **022**, execute the **umask 022** command.

Linux platforms and Solaris operating environments

On Linux platforms or Solaris operating environments, make sure that the `/etc` directory contains a shadow password file. The shadow password file is named shadow and is in the `/etc` directory. If the shadow password file does not exist, an error occurs after enabling global security and configuring the user registry as local operating system.

To create the shadow file, run the pwconv command (with no parameters). This command creates an `/etc/shadow` file from the `/etc/passwd` file. After creating the shadow file, you can configure local operating system security.

Windows platforms

On Windows platforms, the logon user must be a member of the administrator group with the rights of **Act as part of the operating system** and **Log on as a service**.

To add the rights to a user on a Windows 2000 platform:

1. Click **Start** > **Programs** > **Administrative Tools** > **Local Security Policy** (for domain configuration, select **Domain Security Policies**, instead).
2. From the Local Security Settings Panel, click **Local Policies** > **User Rights Assignment** and add the following rights to the user ID:
   - Act as part of the operating system
   - Log on as a service

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 *install_root*`\profiles\`*profile_name*`\config` and *install_root*`\profiles\`*profile_name*`\properties` , except for those in the following list, need protection. For example, give permission to the user who logs onto the system for WebSphere Application Server primary administrative tasks. Other users or groups, such as WebSphere Application Server console users and console groups, who perform partial WebSphere Application Server administrative tasks, like configuring, starting servers and stopping servers, need permissions as well.

The files in the *install_root*`\profiles\`*profile_name*`\properties` directory that should not be protected are:

- `TraceSettings.properties`
- `client.policy`
- `client_types.xml`
- `implfactory.properties`
- `sas.client.props`
- `sas.stdclient.properties`
- `sas.tools.properties`
- `soap.client.props`
- `wsadmin.properties`
- `wsjaas_client.conf`

1. Secure files on a Windows system:
   a. Open the browser for a view of the files and directories on the machine.
   b. Locate and right-click the file or the directory that you want to protect.
   c. Click **Properties**.
   d. Click the **Security** tab.
   e. Remove the `Everyone` entry and any other user or group that you do not want to have access to the file.
   f. Add the users who can access the files with the proper permission.
2. Secure files on UNIX systems. This procedure applies only to the ordinary UNIX file system. If your site uses access-control lists, secure the files by using that mechanism. Any site-specific requirements can affect the owner, group, and corresponding privileges. For example, on AIX,
   a. Go to the *`install_root`* directory and change the ownership of the directory configuration and properties to the user who logs onto the system for WebSphere Application Server primary administrative tasks. Run the following command: `chown -R` *`logon_name directory_name`*

      Where:

      - *login_name* is a specified user or group.
      - *directory_name* is the name of the directory that contains the files.

      It is recommended that you assign ownership of the files that contain password information to the user who runs the application server. If more than one user runs the application server, provide permission to the group in which the users are assigned in the user registry.
   b. Set up the permission by running the following command: `chmod -R 770` *`directory_name`*.
   c. Go to the *`install_root`*`\profiles\`*`profile_name`*`\properties` directory and set the following file permission to **everybody** by running the following command: `chmod 777` *`file_names`*. where *`file_names`* are the following files:
      - `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`
   d. Create a group for WebSphere Application Server and put the users who perform full or partial WebSphere Application Server administrative tasks in that group.
   e. If you want to use WebSphere MQ as a JMS provider, restrict access to the `/var/mqm` directories and log files used. Give write access to the user ID mqm or members of the mqm user group only.

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

The WebSphere Application Server has several plain text passwords. These passwords are not encrypted, but are encoded. The following is a list of files with encoded passwords:

Important: *WAS_INSTALL_ROOT* is a WebSphere Application Server Environment variable that you can configure through the administrative console by clicking **Environment > WebSphere variables**.

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 file password** • **Trust file password** • **Cryptographic token device password**
`WAS_INSTALL_ROOT \profiles \profile_name\properties \sas.client.props`	Specifies passwords for: • com.ibm.ssl.keyStorePassword • com.ibm.ssl.trustStorePassword • com.ibm.CORBA.loginPassword
`war/WEB-INF/ibm_web_bnd.xml`	Specify passwords for the default basic authentication for the ″resource-ref″ bindings within all descriptors (except in the Java cryptography architecture)
`ejb jar/META-INF/ibm_ejbjar_bnd.xml`	Specify passwords for the default basic authentication for the ″resource-ref″ bindings within all descriptors (except in the Java crytography architecture)
`client jar/META-INF/ibm-appclient_bnd.xml`	Specify passwords for the default basic authentication for the ″resource-ref″ bindings within all descriptors (except in the Java crytography architecture)
`ear/META-INF/ibm_application_bnd.xml`	Specify passwords for the default basic authentication for the ″run as″ bindings within all descriptors
`WAS_INSTALL_ROOT \profiles\profile_name\config\cells \cell_name\nodes\node_name\servers \server.xml`	The following fields contain encoded passwords: • **Key file password** • **Trust file password** • **Cryptographic token device password** • **Authentication target password** • **Session persistence password** • **DRS Client data replication password**

File name	Additional information
WAS_INSTALL_ROOT\profiles\profile_name\config\cells \cell_name\nodes\node_name \servers\server1\resources.xml	The following fields contain encoded passwords: • **WAS40Datasource password** • **mailTransport password** • **mailStore password** • **MQQueue queue mgr password**
For WebSphere Application Server and WebSphere Application Server Express: • *WAS_INSTALL_ROOT*\profiles\profile_name\config\cells \cell_name\ws-security.xml • *WAS_INSTALL_ROOT*\profiles\profile_name\config\cells \cell_name\nodes\node_name\servers\server1\ws-security For Network Deployment: *WAS_INSTALL_ROOT*\profiles\profile_name\config\cells \cell_name\ws-security.xml	
ibm-webservices-bnd.xmi	
ibm-webservicesclient-bnd.xmi	
WAS_INSTALL_ROOT \profiles\profile_name \properties\soap.client.props com.ibm.ssl.trustStorePassword	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
WAS_INSTALL_ROOT \profiles\profile_name\properties \sas.stdclient.propertiescom.ibm.ssl.keyStorePassword	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 SAS properties files, type PropFilePasswordEncoder *file_name* -sas and the PropFilePasswordEncoder file encodes the known SAS properties.

   If you are encoding files that are not 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.

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.

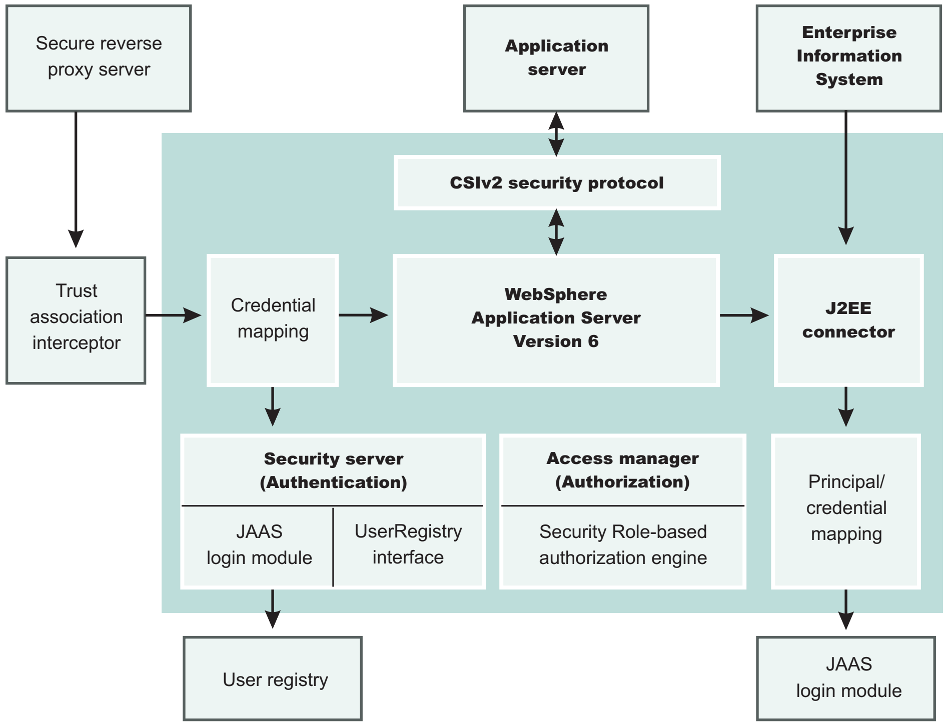
The following examples demonstrate the correct syntax.

```
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 J2EE specifications wherever applicable. WebSphere Application Server 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:


Secure reverse proxy server
Application server
Enterprise Information System
CSIv2 security protocol
Trust association interceptor
Credential mapping
WebSphere Application Server Version 6
J2EE connector
Security server (Authentication)
JAAS login module
UserRegistry interface
Access manager (Authorization)
Security Role-based authorization engine
Principal/ credential mapping
User registry
JAAS login module

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.

WebSphere Application Server plug-in
Redirects all the requests for servlets and JSP pages. Also referred to in WebSphere Application Server literature as *Web server redirector* was introduced to separate Web server from 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 note from the Internet.

Directory
Provides information about the users and their rights in the Web 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.

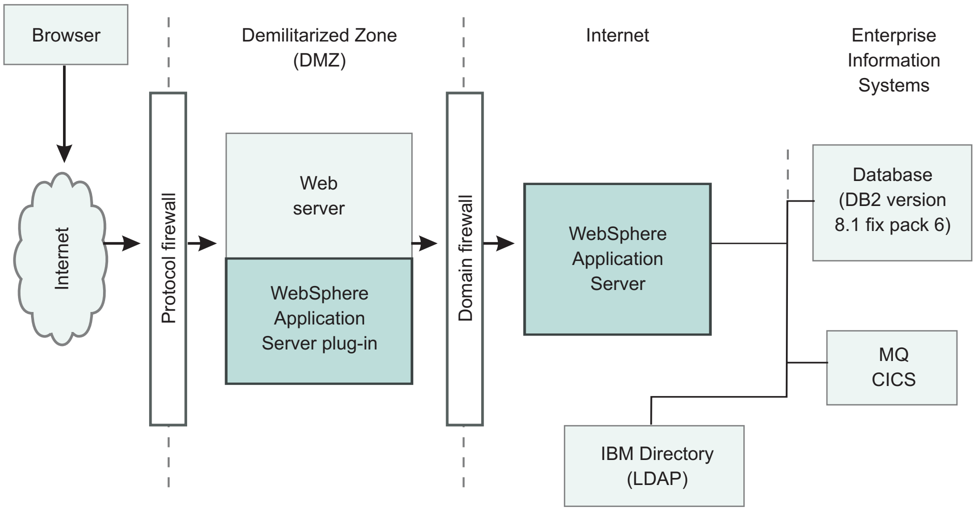
Enterprise information system
Represents existing enterprise applications and business data in back-end databases.

WebSphere Application Server provides the infrastructure to run application business logic and communicate with internal back-end systems and databases 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 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. You can configure the HTTP server to require client certificate authentication with self-signed certificates and to trust only the signer certificate.

For more information, refer to Network communication using Secure Sockets Layer and the Channel Framework

For instructions on how to generate self-signed certificates and how to set up secure communications channels between an HTTP server and the WebSphere Application Server plug-in, refer to Configuring IHS plug-in and the Internal Web Server for SSL and Configuring IHS for SSL Mutual Authentication.


Browser
Demilitarized Zone (DMZ)
Internet
Enterprise Information Systems
Internet
Protocol firewall
Web server
WebSphere Application Server plug-in
Domain firewall
WebSphere Application Server
Database (DB2 version 8.1 fix pack 6)
MQ CICS
IBM Directory (LDAP)

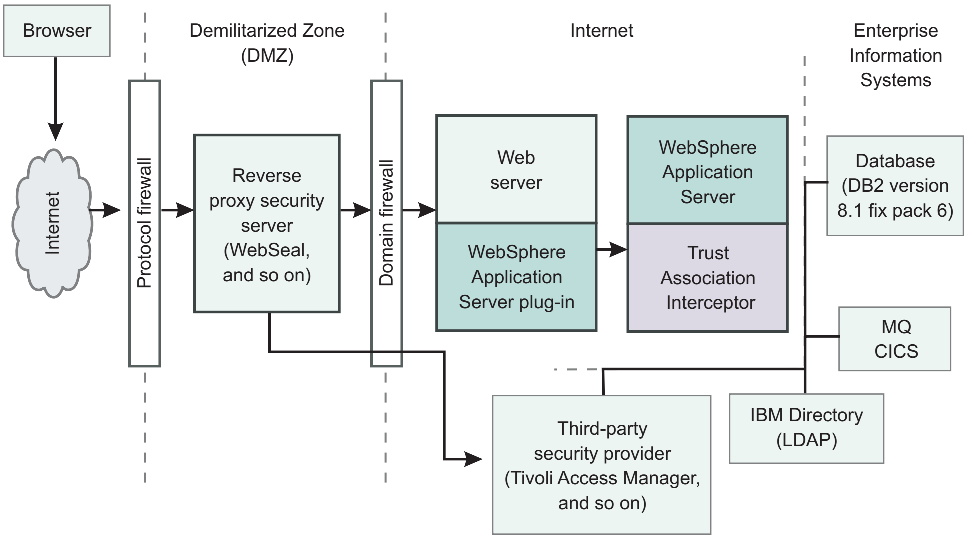
The WebSphere Application Server plug-in routes HTTP requests according to the virtual host and port configuration and the URL pattern matching. Client authentication and finer grained access control are handled by WebSphere Application Server behind the firewall.

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 as illustrated in the picture. WebSphere Application Server supports various user registry implementations through the pluggable user registry interface.

WebSphere Application Server ships a Local OS user registry implementation for Windows, AIX, AS/400, and Lightweight Directory Access Protocol (LDAP).

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 Light Weight 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 forward 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, Configuring trust association interceptors.


Browser
Demilitarized Zone (DMZ)
Internet
Enterprise Information Systems
Internet
Protocol firewall
Reverse proxy security server (WebSeal, and so on)
Domain firewall
Web server
WebSphere Application Server plug-in
WebSphere Application Server
Trust Association Interceptor
Database (DB2 version 8.1 fix pack 6)
MQ CICS
IBM Directory (LDAP)
Third-party security provider (Tivoli Access Manager, and so on)

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 not passed in plain text. To set up an SSL connection from WebSphere Application Server to the LDAP server, refer to Configuring SSL for the LDAP client. WebSphere Application Server Version 5 supports the J2EE Connector Architecture (JCA). 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 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.

Each connector can be configured to use a different set of IDs and passwords. For a description on how to configure JCA principal mapping user IDs and passwords, refer to Managing J2C Authentication Data Entries. 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 Java 2 security mapping module underneath JAAS Programmatic Login and ″Managing Java Authentication and Authorization Service (JAAS) Login Configuration.″

Security and WebSphere MQseries

It is important to note that security logging information on UNIX systems is not protected because of the world-writeable files in the `/var` file system of MQseries. MQseries ships the following files with its product:

- -rw-rw-rw- /var/mqm/errors/AMQERR01.LOG
- -rw-rw-rw- /var/mqm/errors/AMQERR02.LOG
- -rw-rw-rw- /var/mqm/errors/AMQERR03.LOG

The previously mentioned files are world-writeable and enable any users on the system to fill up the /var file system where all the security logging information is stored. This leaves the security information unprotected because anyone can access the logging information without being tracked.

To work around this problem, create a file system for the embedded messaging component working data on UNIX. Before you install the embedded messaging component of WebSphere Application Server on UNIX platforms, consider creating and mounting a journalized file system called /var/mqm. Use a partition strategy with a separate volume for the WebSphere MQ data. This means that other system activity is not affected if a large amount of WebSphere MQ work builds up.

To determine the size of the /var/mqm file system for a server installation, consider the following:
- Maximum number of messages in the system at one time
- Contingency for message buildups, if there is a system problem
- Average size of the message data, plus 500 bytes for the message header
- Number of queues
- Size of log files and error messages

Allow 50MB as a minimum for a WebSphere MQ server. You need less space in the /var/mqm file system for a WebSphere MQ client (typically 15MB).

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).

Interoperating with a C++ common object request broker architecture client

You can achieve interoperability of Security Authentication Service between the C++ Common Object Request Broker Architecture (CORBA) client and WebSphere Application Server using Common Secure Interoperability Version 2 (CSIv2) authentication protocol over Remote Method Invocation over the Internet Inter-ORB Protocol (RMI-IIOP). The CSIv2 security service protocol has authentication, attribute and transport layers. Among the three layers, transport authentication is conceptually simple, however, cryptographically based transport authentication is the strongest. WebSphere Application Server Enterprise has implemented the transport authentication layer, so that C++ secure CORBA clients can use it effectively in making CORBA clients and protected enterprise bean resources work together.

Security authentication from non-Java based C++ client to enterprise beans. WebSphere Application Server supports security in the CORBA C++ client to access protected enterprise beans. If configured, C++ CORBA clients can access protected enterprise bean methods using client certificate to achieve mutual authentication on WebSphere Application Server Enterprise 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 WAS_CONFIG_FILE=/WebSphere/V5R0M0/AppServer/bin/current.env.

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.

To support the C++ CORBA client in accessing protected enterprise beans:

1. Obtain a valid certificate to represent the client and export its public key to the target enterprise bean server.

   A valid certificate is needed to represent the C++ client. Request a certificate from the certificate authority (CA) or create a self-signed certificate for testing purposes.

   Use the Key Management Utility from the Global Security Kit (GSKit) to extract the public key from the personal certificate and save it in the `.arm` format. For details, see the related information about how to extract the personal certificate of the public key.

2. Prepare a truststore file for WebSphere Application Server.

   Add the extracted client public key in the `.arm` file from the client to the server key truststore file. The server can now authenticate the client.

   Note: This is done by invoking the Key Management Utility through ikeyman.bat or ikeyman.sh from WebSphere Application Server installation.

   For details, see the article on Adding truststore files.

3. Configure WebSphere Application Server to support SSL as the authentication mechanism.

   a. Start the administrative console.

   b. Locate the application server that has the target enterprise bean deployed and configure it to use SSL client certificate authentication.

      If it is a base installation, complete the following steps:

      1) Click **Security > Global security**. Under Authentication, click **Authentication protocol > CSIv2 inbound authentication**. Select **Supported** for the Basic authentication and Client certificate authentication options. Leave the rest of the options as defaults.

      2) Click **Security > Global security**. Under Authentication, click **Authentication protocol > CSIv2 inbound transport** and verify that the **SSL-supported** option is selected.

      If it is a Network Deployment setting, complete the following steps:

      1) Click **Server > Application Server >***server_name_where_the_EJB_resides*. Under security, click **Server security**. Under Additional properties, click **CSI inbound authentication**. Select **Supported** for the Basic authentication and Client certificate authentication options. Leave the rest of the options as defaults.

      2) Click **Server > Application Server >***server_name_where_the_EJB_resides*. Under security, click **Server security**. Under Additional properties, click **CSI inbound transport**. Verify that the **SSL-Supported** option is selected.

      For details, see the security articles Configuring CSIv2 inbound authentication and Configuring CSIv2 inbound transport.

   c. Restart the application server.

      The WebSphere Application Server is ready to take a C++ CORBA security client and a mutually authenticated server and client by using SSL in the transport layer.

4. Configure the C++ CORBA client to use a certificate in performing the mutual authentication.

   Client users are accustomed to using property files in their applications because they are helpful in specifying configuration settings. The following list presents important C++ security settings:

C++ security setting	Description
com.ibm.CORBA.bootstrapHostName=ricebella.austin.ibm.com	Specifies the target host name.

com.ibm.CORBA.securityEnabled=yes	Enables security.
com.ibm.CSI.performTLClientAuthenticationSupported=yes	Ensures client is supporting mutual authentication by certificate
com.ibm.CSI.performTransportAssocSSLTLSSupported=yes	Ensures SSL is used, not TCP/IP
com.ibm.ssl.keyFile=C:/ricebella/etc/DummyKeyRingFile.KDB	Specifies which key database file to use.
com.ibm.ssl.keyPassword=WebAS	Specifies the password for opening the key database file. WebSphere Application Server supports a utility called `PasswordEncode4cpp` to encode the plain password.
com.ibm.CORBA.translationEnabled=1	Enables the valueType conversion.

To use the property files in running a C++ client, an environment variable WASPROPS, is used to indicate where a property file or a list of property files exist.

For the complete set of C++ client properties, see the sample property file `scclient.props`, which is shipped with the product located in the *install_root*`\profiles\`*profile_name*`\etc` directory.

Interoperating with previous product versions

IBM WebSphere Application Server, Version 5.x or later interoperates with the previous product versions (such as Version 4 and Version 3.5). Interoperability is achieved only when the Lightweight Third Party Authentication (LTPA) authentication mechanism and Lightweight Directory Access Protocol (LDAP) user registry are used. Credentials derived from Simple WebSphere Authentication Mechanisms (SWAM) are not forwardable.

1. Enable security with the LTPA authentication mechanism and the LDAP user registry. Make sure that the same LDAP user registry is shared by all the product versions.
2. Extract and add Version 5 server certificates into the server key ring file of the previous version.
   a. Open the Version 5 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 product Version 5.
3. Extract and add Version 5 server certificates into the server key ring file of the previous version.
   a. Open the Version 5 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 product Version 5.
4. Extract and add Version 5 trust certificates into the trust key ring file of the previous product version.
   a. Open the Version 5 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 Version 5.
5. If single signon (SSO) is enabled, export keys from the Version 5 product and import them into the previous product version. The Version 4 product requires the fix, PQ61779, and the Version 3.5 product requires the fix, PQ59667, for SSO to function.
6. Verify that the application uses the correct JNDI name. In Version 5, 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 5 enterprise beans.

   You can also create EJB name bindings in Version 5 that are compatible with the previous version. To create an EJB name binding at the root Version 5, 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 5, 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 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 specifications"
- "Administration" on page 1417

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

Programming model and decisions

- JSSE Documentation.

  Refer to the http://www.ibm.com/developerworks/java/jdk/security/jsseDocs.zip file for the Javadoc of the APIs, JSSE Reference Guide, and JSSE samples.
- iKeyman documentation.

  Look in the http://www.ibm.com/developerworks/java/jdk/security/iKeymanDocs.zip file for the Secure Sockets Layer (SSL) Introduction and iKeyman documentation.
- JCE documentation.
  - For the JCA spec and JCE API usage refer to the http://www.ibm.com/developerworks/java/jdk/security/jceDocs.zip file.
  - For JCE sample applications refer to the http://www.ibm.com/developerworks/java/jdk/security/jceDocs.zip file.
  - For Java Cryptography Architecture Reference refer to the http://www.ibm.com/developerworks/java/jdk/security/jceDocs.zip file.
  - For how to implement a JCE provider refer to the http://www.ibm.com/developerworks/java/jdk/security/jceDocs.zip file.
  - For the Javadoc of JCE APIs refer to the http://www.ibm.com/developerworks/java/jdk/security/jceDocs.zip file.
  - For the 1.4.2 release of the IBM developer kit for the Java platform refer to the http://www-106.ibm.com/developerworks/java/library/j-ibmsecurity.html file.

Programming specifications

- J2EE Specifications
- EJB Specifications
- Servlet Specifications

- Common Secure Interoperability Version 2 (CSIv2) Specification
- JAAS Specification.

  For programming and usage in JAAS, refer to the specification located at http://www.ibm.com/developerworks/java/jdk/security/ and scroll down to find the JAAS documentation for your platform. This document contains the following when unpacked:
  – login.html - LoginModule Developer's Guide
  – api.html - Developer's Guide (JAAS JavaDoc)
  – HelloWorld.tar - Sample JAAS Application
- Java 2 Platform, Standard Edition, v 1.4.2 API Specification
- Java Authorization Contract for Containers (JSR 115) Specification

Administration
- WebSphere Application Server Version 4.0 Security Redbook: WebSphere Security Model.
- 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
- WebSphere education on demand: Enabling security best practice tutorials

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 1418.
2. Assign users to administrator roles. For more information, see “Assigning users to administrator roles” on page 1430.
3. Assign users to naming roles. For more information, see “Assigning users to naming roles” on page 1432.
4. Configure authentication mechanisms. For more information, see “Configuring authentication mechanisms” on page 1436.
5. Configure Lightweight Third Party Authentication. For more information, see “Configuring Lightweight Third Party Authentication” on page 1437.
6. Configure trust association interceptors. For more information, see “Configuring trust association interceptors” on page 1445.
7. Configure single signon. For more information, see “Configuring single signon” on page 1448.
8. Configure user registries. For more information, see “Configuring user registries” on page 1466.
   a. Configure local operating system user registries. For more information, see “Configuring local operating system user registries” on page 1471.
   b. Configure Lightweight Directory Access Protocol user registries. For more information, see “Configuring Lightweight Directory Access Protocol user registries” on page 1474.
   c. Configure custom user registries. For more information, see “Configuring custom user registries” on page 1488.
9. Configure Java Authentication and Authorization Service login. For more information, see ″Configuring application logins for Java Authentication and Authorization Service″ in the information center.
10. Configure an authorization provider. For more information, see “Configuring a JACC provider” on page 1617. 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 1627 or “Configuring the JACC provider for Tivoli Access Manager using the administrative console” on page 1629.

11. 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 1657.
12. Configure Secure Sockets Layer. For more information, see ″Configuring Secure Sockets Layer″ in the information center.
13. Configure Java 2 Security Manager. For more information, see “Configuring Java 2 security” on page 1740.
14. **Optional:** Configure security attribute propagation. For more information, see “Security attribute propagation” on page 1551.

Global security

Global security applies to all applications running in the environment and determines whether security is used at all, the type of registry against which authentication takes place, and other values, many of which act as defaults.

The term *global security* represents the security configuration that is effective for the entire security domain. A *security domain* consists of all servers configured with the same user registry *realm* name. In some cases, the realm can be the machine name of a Local OS user registry. In this case, all application servers must reside on the same physical machine. In other cases, the realm can be the machine name of an Lightweight Directory Access Protocol (LDAP) user registry. Since LDAP is a distributed user registry, a multiple node configuration is supported, such as the case for a Network Deployment environment. The basic requirement for a security domain is that the access ID returned by the registry from one server within the security domain is the same access ID as that returned from the registry on any other server within the same security domain. The *access ID* is the unique identification of a user and is used during authorization to determine if access is permitted to the resource.

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 you can configure includes Java 2 Security Manager, Java Authentication and Authorization Service (JAAS), Java 2 Connector authentication data entries, Common Secure Interoperability Version 2 (CSIv2)/Security Authentication Service (SAS) authentication protocol (Remote Method Invocation over the Internet Inter-ORB Protocol (RMI/IIOP) security), and other miscellaneous attributes. The global security configuration usually applies to every server within the security domain.

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”
- Introduction: Security

After you understand the security components, you can proceed to configure global security in WebSphere Application Server.

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** and configure the authentication mechanism, user registry, and so on. The configuration order is not important. However, when you select the **Enable global security** flag in the **Global security** panel, verify that all these tasks are completed. When you 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 you do not configure these, the validation fails.
3. Configure a user registry. For more information, see “Configuring user registries” on page 1466. You can configure a local OS, LDAP, or custom user registry through the links under User registry on the Global security panel.

   One of the details common to all user registries is the **server user ID**. 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 server user ID can access all protected administrative methods. On Windows systems, the ID must not be the same name as the machine name of your system, since the registry sometimes returns machine-specific information when querying a user of the same name. In LDAP user registries, verify that the server user ID is a member of the registry and not just the LDAP administrative role ID. The entry must be searchable.

   The server user ID does **not** run WebSphere Application Server processes. Rather, the process ID runs the WebSphere Application Server processes.The process ID runs the WebSphere Application Server processes.

   The process ID is determined by the way the process starts. For example, if you use a command line to start processes, the user ID that is logged into the system is the process ID. If running as a service, the user ID that is logged into the system is the user ID running the service. If you choose the LocalOS registry, the process ID requires special privileges to call the operating system APIs. Specifically, the process ID must have the **Act as Part of Operating System** privileges on Windows systems or **root** privileges on a UNIX system.
4. Configure the authentication mechanism. To get details about configuring authentication mechanisms, read the “Configuring authentication mechanisms” on page 1436 article. There are two authentication mechanisms to choose from in the Global Security panel: Simple WebSphere Authentication Mechanism (SWAM) and Lightweight Third-Party Authentication (LTPA). However, only LTPA requires any additional configuration parameters. Use the SWAM option for single server requirements. Use the LTPA option for multi-server distributed requirements. SWAM credentials are not forwardable to other machines and for that reason do not expire. Credentials for LTPA are forwardable to other machines and for security reasons do expire. This expiration time is configurable. If you choose to go with LTPA, then “Configuring single signon” on page 1448 support. This support permits browsers to visit different product servers without having to authenticate multiple times.
5. Configure the authentication protocol for special security requirements from Java clients, if needed. This task entails choosing a protocol, either Common Secure Interoperability Version 2 (CSIv2) or Security Authentication Service (SAS). The SAS protocol is still provided as a backwards compatibility to previous product releases, but is being deprecated. For details on configuring CSIv2 or SAS, see the article, “Configuring Common Secure Interoperability Version 2 and Security Authentication Service authentication protocols” on page 1657.
6. Modify the default Secure Sockets Layer (SSL) keystore and truststore files that are packaged with the product. This action protects the integrity of the messages sent across the Internet. The product provides a single location where you can specify SSL configurations that the various WebSphere Application Server features that use SSL can utilize, including the LDAP user registry, Web container and the authentication protocol (CSIv2 and SAS). Create a new keystore and truststore, by referring to the “Creating a keystore file” on page 1717 and “Creating truststore files” on page 1721 articles. You can create different keystore files and truststore files for different uses or you can create just one set for everything that the server uses Secure Sockets Layer (SSL) for. After you create these new keystore and truststore files, specify them in the **SSL Configuration Repertoires**. To get to the **SSL Configuration Repertoires**, click **Security > SSL**. See the article, ″Configuring Secure Sockets Layer″ in the information center. To get to the SSL Configuration Repertoire, click **Security > SSL**. You can either edit the DefaultSSLConfig file or create a new SSL configuration with a new alias name. If

you create a new alias name for your new keystore and truststore files, change every location that references the DefaultSSLConfig SSL configuration alias. The following list specifies the locations of where the SSL configuration repertoire aliases are used in the WebSphere Application Server configuration.

For any transports that use the new network input/output channel chains, including HTTP and Java Message Service (JMS), you can modify the SSL configuration repertoire aliases in the following locations for each server:

- Click **Server > Application server >** *server_name*. Under Communications, click **Ports**. Locate a transport chain where SSL is enabled and click **View associated transports**. Click *transport_channel_name*. Under Transport Channels, click **SSL Inbound Channel (SSL_2)**.

For the Object Request Broker (ORB) SSL transports, you can modify the SSL configuration repertoire aliases in the following locations. These configurations are for the server-level for WebSphere Application Server and WebSphere Application Server Express and the cell level for WebSphere Application Server Network Deployment.

- Click **Security > 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 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 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**.

7. Click **Security** > **Global security** to configure the rest of the security settings and 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. See the “Global security settings” on page 1421 article for detailed information about these fields. When you complete all of the fields, click **OK** or **Apply** to accept the selected settings. Click **Save** to persist these settings out to a file. If you see any informational messages in red text color, then a problem has occurred with the security validation. Typically, the message indicates the problem; therefore, review your configuration to verify that the user registry settings are accurate and the correct registry is selected. In some cases the LTPA configuration might not be fully specified.
8. Store the configuration for the server to use after it restarts. Complete this action if you have clicked **OK** or **Apply** on the **Security > Global security** panel, and there are no validation problems. To save the configuration, click **Save** in the menu bar at the top. This action writes the settings out to the configuration repository. If you do not click **Apply** or **OK** in the **Global security** panel before clicking **Save** on the main menu, your changes are not written to the repository.
9. 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 ID and password, which is typically the server user ID specified when you configure the user registry.

Enabling global security:

You can decide whether to enable IBM WebSphere Application Server security. You must enable security for all other security settings to function.

Note: WebSphere Application Server uses cryptography to protect sensitive data and ensure confidentiallity 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 JSSE and JCE libraries in the JDK to enforce this cryptography. The JDK provides strong but limited jurisdiction policy files. Unrestricted policy files provide the ability to perform full strength cryptography and improve performance.

IBM's SDKs ship with strong but limited jurisdiction policy files. For Windows, Linux, HPUX Solaris, and AIX platforms, the unrestricted policy files can be downloaded from https://www6.software.ibm.com/dl/jcesdk/jcesdk-p (Use the link near the bottom of the page.) The ZIP file should be unpacked and the two JAR files placed in the `JRE's jre/lib/security/` directory.

1. Select the unrestricted JCE policy files for SDK 1.4.2.
2. Extract the unlimited jurisdiction policy files that are packaged in the ZIP file that contains two files:
   a. `US_export_policy.jar`
   b. `local_policy.jar`
3. Back up the original version of the policy files.
4. Replace the policy files in the `$JAVA_HOME/jre/lib/security` directory with the two files above.

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**.

When security is disabled, WebSphere Application Server performance is increased between 10-20%. Therefore, consider disabling security when it is not needed.

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.

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.

If your server does not restart after you enable global security, you can disable security. Go to your `$install_root\bin` directory and execute 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.

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.

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. Security information pertaining to beans, permissions, and credentials is cached. When the cache timeout expires, all cached information becomes invalid. Subsequent requests for the information result in a database lookup. Sometimes, acquiring the information requires invoking a Lightweight Directory Access Protocol

(LDAP)-bind or native authentication. Both invocations are relatively costly operations for performance. Determine the best trade off for the application, by looking at usage patterns and security needs for the site.

In a 20-minute performance test, setting the cache timeout so that a timeout does not occur yields a 40% performance improvement.

Data type:	Integer
Units:	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.

In previous releases the Security Authentication Service (SAS) protocol 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 servers, specify `CSI` as your protocol.

If some servers are version 3.x or version 4.x servers, specify `CSI and SAS`.

Default:	BOTH
Range:	CSI and SAS, CSI
Range:	

Active authentication mechanism:

Specifies the active authentication mechanism when security is enabled.

WebSphere Application Server and WebSphere Application Server Express, Version 6 support the following authentication mechanisms: Simple WebSphere Authentication Mechanism (SWAM) and Lightweight Third Party Authentication (LTPA).

Default:	SWAM (WebSphere Application Server)
Default:	
Range:	SWAM, LTPA

Default:	
Range:	

Active User Registry:

Specifies the active user registry, when security is enabled. LDAP or a custom user registry is required when running as a UNIX nonroot user or running in a multi-node environment.

You can configure settings for one of the following user registries:

- Local OS

  When you enable global security on a UNIX platform and the user registry is the local OS, you must run the server as root. The local OS user registry is not supported for nonroot users on a UNIX platform.
- 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 use of Federal Information Processing Standard (FIPS)-approved cryptographic algorithms.

When **Use the Federal Information Processing Standard (FIPS)** is enabled, 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. If you select the **Use the Federal Information Processing Standard (FIPS)** option prior to specifying a FIPS-approved JSSE provider and a TLS protocol, the following error message displays at the top of the **Global security** panel:

```
The security policy is set to use only FIPS approved cryptographic algorithms.
However at least one SSL configuration may not be using a FIPS approved JSSE provider.
FIPS approved cryptographic algorithms may not be used in those cases.
```

To correct this problem, configure your JSSE provider and security protocol on the **SSL configuration repertoires** panel by completing one of the following tasks:

- Clicking **Security > SSL** and modifying an existing configuration
- Clicking **New** and creating a new configuration

Note:

- The IBMJSSEFIPS provider is not supported on the HP-UX platform.
- In WebSphere Application Server Version 6, the HTTP transport does not support the FIPS-approved providers because it uses the new IBMJSSE2 provider to support channel framework. The channel framework supports asynchronous communication that might enhance performance. The IBMJSSE2 provider is specified for the Secure Sockets Layer (SSL) channel because it must use the IBMJCE provider for encryption. Currently, an IBMJCEFIPS provider that works with the IBMJSSE2 provider, is not available.

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**.

Configuring server security

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 Secure Authentication Service (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 in 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**
- **SAS Inbound Transport**
- **SAS Outbound Transport**
- **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**

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 1418 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 SAS have not been overridden at the server level. CSI and 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 **Enabled** flag 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 1657 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-level security settings:

Use this page to enable server level security and specify other server level security configurations.

To view this administrative console page, click **Servers > Application Servers >** *server_name*. Under Security, click **Server Security**. 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.

If some servers are Version 3.x or Version 4.x servers, it is best to specify **CSI and SAS**. However, by specifying **CSI and SAS**, you now have two interceptors invoking each request.

Data type	String
Default	CSI and SAS

Range	CSI, CSI and SAS

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.

The server identity specified when enabling global security is automatically mapped to the administrative 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.

When global security is enabled, WebSphere Application Servers run under the server identity that is defined under the active user registry configuration. Although it is not shown on the administrative console and in other tools, a special Server subject is mapped to the administrator role. The WebSphere Application Server run-time code, which runs under the server identity, requires authorization to runtime operations. If no other user is assigned administrative roles, you can log into the administrative console or to the wsadmin scripting tool using the server identity to perform administrative operations and to assign other users or groups to administrative roles. Because the server identity is assigned to the administrative role by default, the administrative security policy requires the administrative role to perform the following operations:

- Change server ID and server password
- Enable or disable WebSphere Application Server global security
- Enforce or disable Java 2 Security

- Change the LTPA password or generate keys
- Assign users and groups to administrative roles

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 JNDI interfaces. The second is with CORBA clients invoking CosNaming methods directly.

Four security roles are introduced :

- CosNamingRead
- CosNamingWrite
- CosNamingCreate
- CosNamingDelete

The names of the four roles are the same with WebSphere Application Server Advanced Edition Version 4.0.2. The roles now 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.

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 (enabled with PQ81586) .

Users, groups, or the special subjects AllAuthenticated and Everyone can be added or removed to or from the naming roles from the WebSphere Application Server administrative console at any time. However, a server restart is required for the changes to take effect. A best practice is to map groups or one of the special-subjects, rather than specific users, to naming roles because it is more flexible and easier to administer in the long run. By mapping a group to a naming role, adding or removing users to or from the group occurs outside of WebSphere Application Server and does not require a server restart for the change to take effect.

The CosNaming authorization policy is only enforced when 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.

In WebSphere Application Server Version 4.0.2, each CosNaming function is assigned to only one role. Therefore, users who are assigned the CosNamingCreate role cannot query the name space unless they have also been assigned CosNamingRead. And in most cases a creator needs to be assigned three roles: CosNamingRead, CosNamingWrite, and CosNamingCreate. The CosNamingRead and CosNamingWrite roles assignment for the creator example are included in the CosNamingCreate role. In most of the cases, WebSphere Application Server administrators do not have to change the roles assignment for every user or group when they move to this release from a previous one.

Although the ability exists to greatly restrict access to the name space by changing the default policy, unexpected `org.omg.CORBA.NO_PERMISSION` exceptions can occur at 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

The following steps are needed to assign users 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 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), Security Authentication Service (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 LDAP, local OS, or a custom registry. You can set up your user registries without enabling security.

Once you assign users to administrative roles, you must restart the server for the new roles to take effect. However, the administrative resources are not protected until you enable security.

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 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, 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 groups):

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:

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, and 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

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.

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

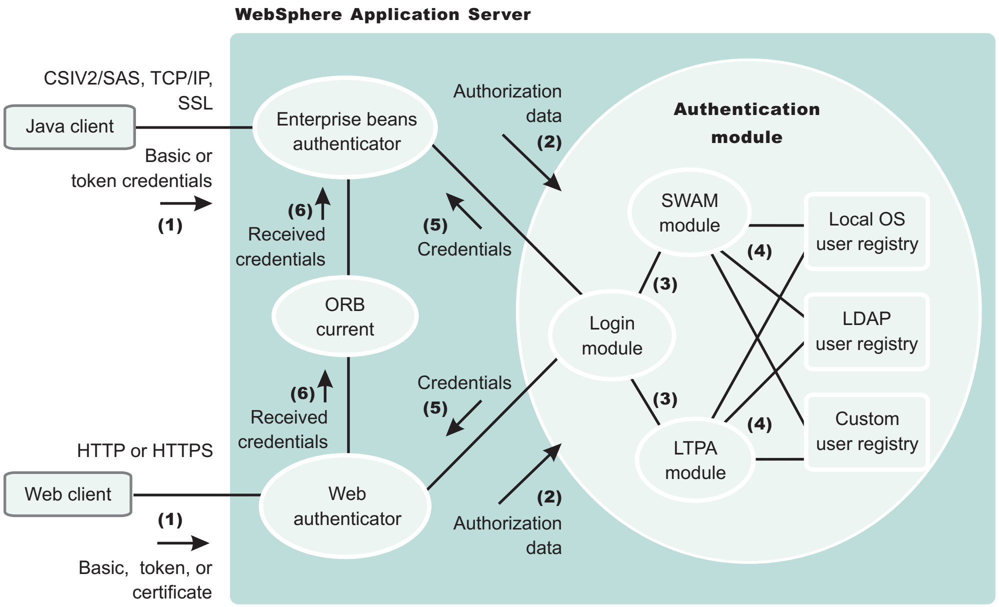
An *authentication mechanism* defines rules about security information (for example, whether a credential is forwardable to another Java process), and the format of how security information is stored in both credentials and tokens.

Authentication is the process of establishing whether a client is 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 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 several 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



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)
- Secure Authentication Service (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 all platforms), 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. 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).

- 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: LocalOS, Lightweight Directory Access Protocol (LDAP), and custom registry. External registry implementation following the registry interface specified by IBM can replace either the LocalOS or the LDAP user registry.

The login module creates a JAAS subject after authentication and stores the Common Object Request Broker Architecture (CORBA) credential derived from the authentication data in the public credentials list of the subject. The credential is returned to the Web authenticator or EJB authenticator (5).

The Web authenticator and the EJB authenticator store the received credentials in the Object Request Broker (ORB) current for the authorization service to use in performing further access control checks.

WebSphere Application Server provides two authentication mechanisms: SWAM and LTPA. These authentication mechanisms differ primarily in the distributed security features that each supports.

Configuring authentication mechanisms

Configure authentication mechanisms by clicking **Authentication Mechanisms** under **Security > Global security** in the administrative console.

For LTPA, follow the steps in “Configuring single signon” on page 1448 for most situations. If trust association is required, follow the steps in “Configuring trust association interceptors” on page 1445.

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.

Since 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.

The Lightweight Third Party Authentication (LTPA) protocol enables the WebSphere Application Server to provide security in a distributed environment using cryptography. Application servers distributed in multiple nodes and cells can securely communicate using this 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 SSO domain are case sensitive and must match identically.

Windows For local OS on the Windows platform, the realm name is the domain name, if a domain is in use, or the machine name.

UNIX On the UNIX platform, the realm name is the same as the host name.

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) or through the authentication layer (Security Authentication Service (SAS) or Common Secure Interoperability Version 2 (CSIv2) for enterprise beans).

If the receiving servers share the same keys as the originating server, the token can be decrypted to obtain the user information, which then is validated to make sure 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 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.

In the base version of WebSphere Application Server, LTPA, ICSF, and the Simple WebSphere Authentication Mechanism (SWAM) protocols are supported.

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
SWAM	No	No	Yes	Yes	Yes
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 mechanisms, click **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.

   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 > Authentication mechanisms > LTPA** in the navigation panel on the left.
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 key propagation is dynamic. All of the processes running at that time (cells, node agents, application servers) are updated with the new set of keys. 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** > **Authentication mechanisms** > **LTPA** in the navigation panel on the left.
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.

Importing keys is a dynamic operation. All of the servers that are running at this time are updated with the new set of keys. Any back-level tokens signed with the back-level keys fail validation and the user is prompted to log in again.

1. Access the administrative console by typing `http://localhost:9060/ibm/console` in a Web browser.
2. Click **Security** > **Authentication mechanisms** > **LTPA** in the navigation panel on the left.
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**.

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

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

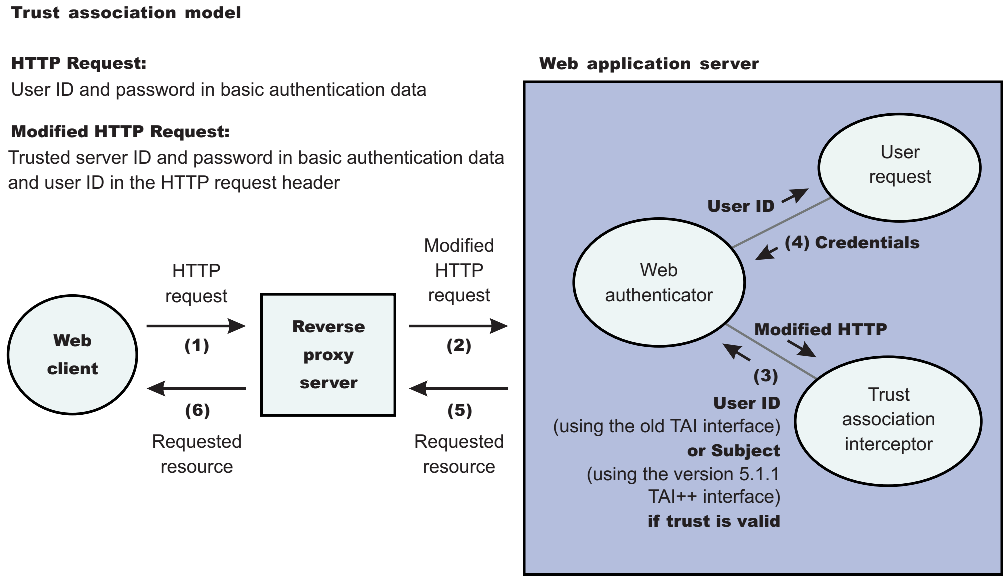
The idea that WebSphere Application Server can support trust association implies that the product application security recognizes and processes HTTP requests received from a reverse proxy server. WebSphere Application Server and the proxy server engage in a contract in which the product gives its full 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.″



IBM WebSphere Application Server: WebSEAL Integration

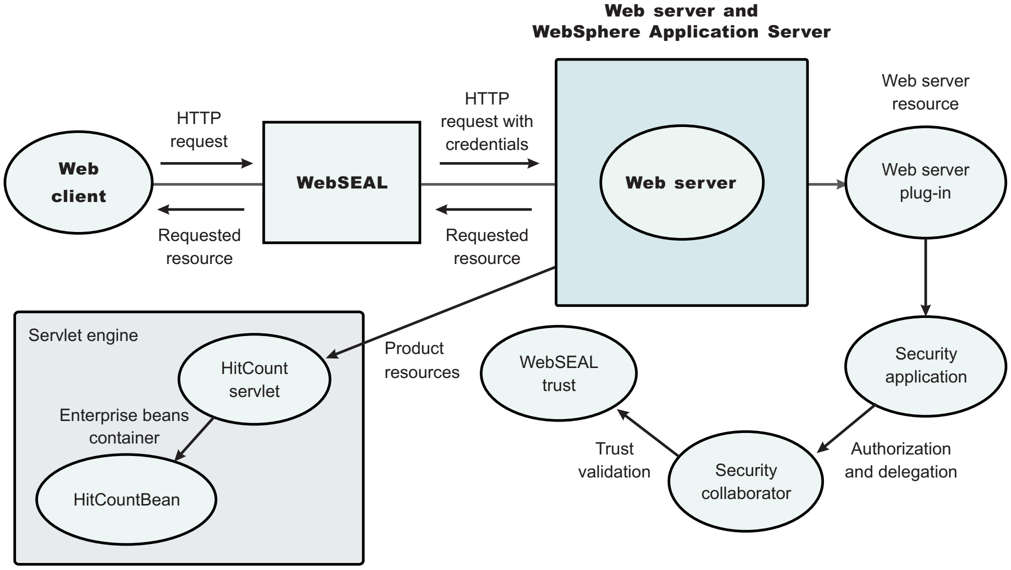
The integration of WebSEAL and WebSphere Application Server security is achieved by placing the WebSEAL server at the front-end as a reverse proxy server. See Figure 2. From a WebSEAL management perspective, a junction is created with WebSEAL on one end, and the product Web server on the other end. A 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

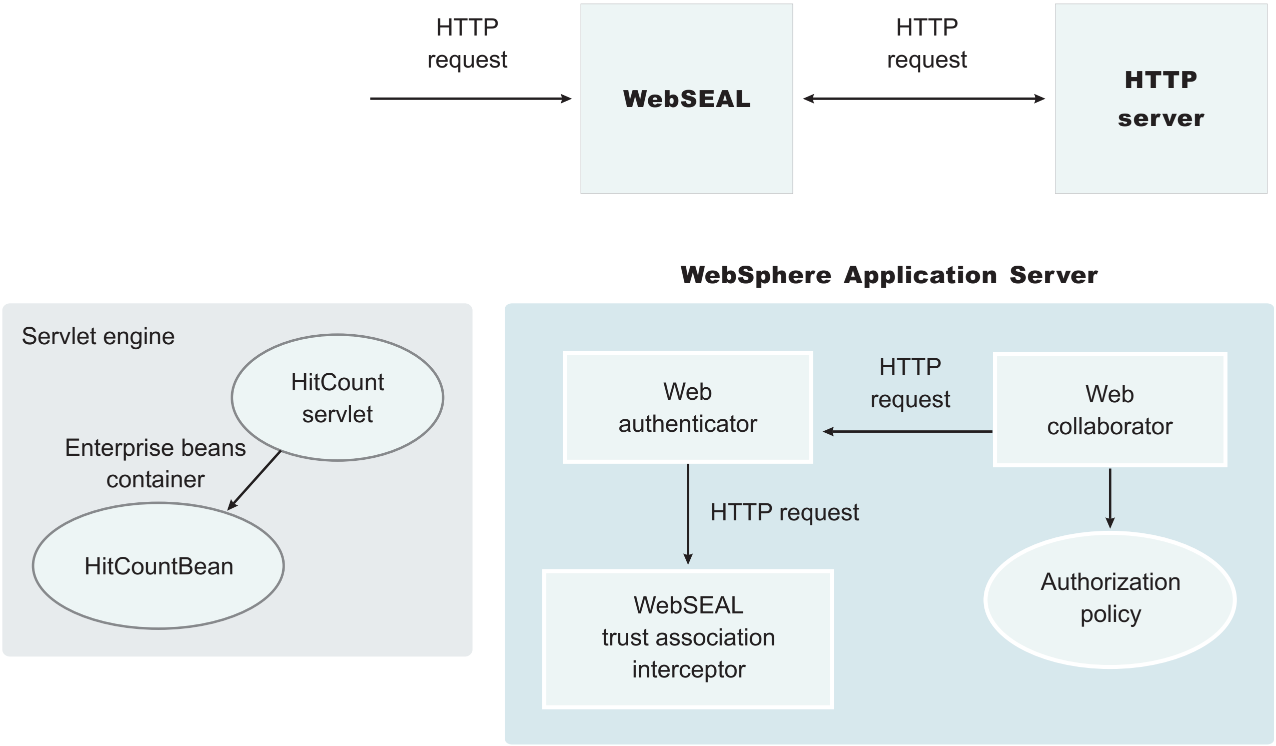
The policy director delegates all of the Web requests to its Web component, the WebSEAL server. One of the major functions of the server is to perform authentication of the requesting user. The WebSEAL server consults a Lightweight Directory Access Protocol (LDAP) directory. It can also map the original 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 interface (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. Refer to X for more information.

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, seeTrust 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 mechanisms, click **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 1457 or “Configuring single signon using trust association interceptor ++” on page 1458. 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.

  This registry can be either a supported Lightweight Directory Access Protocol (LDAP) directory server or, if SSO is configured between two WebSphere Application Servers, a custom 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 only when Lightweight Third Party Authentication (LTPA) is the authentication mechanism.

When SSO is enabled, a cookie is created containing the LTPA token and inserted into the HTTP response. When the user accesses other Web resources in any other WebSphere Application Server process in the same domain name service (DNS) domain, the cookie is sent in the request. The LTPA token is then extracted from the cookie and validated. If the request is between different cells of WebSphere Application Servers, you must share the LTPA keys and the user registry between the cells for SSO to work. The realm names on each system in the SSO domain are case sensitive and must match identically.

Windows For local OS on the Windows platform, the realm name is the domain name if a domain is in use or the machine name.

UNIX On the Linux or UNIX platforms, the release name is the same as the host name.

For the Lightweight Directory Access Protocol (LDAP) the realm name is the host:port realm name of the LDAP server. The LTPA authentication mechanism requires that you enable SSO if any of the Web applications have form login as the authentication method.

Because single signon is a subset of LTPA, it is recommended that you read “Lightweight Third Party Authentication” on page 1436 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″ in the information center.

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 1451.

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.″

   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:

    You can use the OS/400 Qshell, a UNIX shell, or a Windows DOS prompt

    ```
    % 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 1456
3. "Configuring single signon using the trust association interceptor" on page 1457 or "Configuring single signon using trust association interceptor ++" on page 1458

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 1456

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 1455.
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 authorized user’s credentials onto 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 1455. 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 1458

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:

Option	Description
com.ibm.websphere.security.trustassociation.types	Ensure *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 1455 The format of the username is the short name representation. This is a mandatory property. If it is not set in WebSphere then TAI initialization will fail.

com.ibm.websphere.security.webseal.id	The *iv-user* header. That is;
	com.ibm.websphere.security.webseal.id=iv-user

Option	Description
com.ibm.websphere.security.webseal.hostnames	Do not set this property if using Tivoli Access Manager Plug-in for Web Servers. The host names (case sensitive) 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.
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 select, 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:

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 1455 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.
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.

Option	Description

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	Once trust has been 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.

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 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.

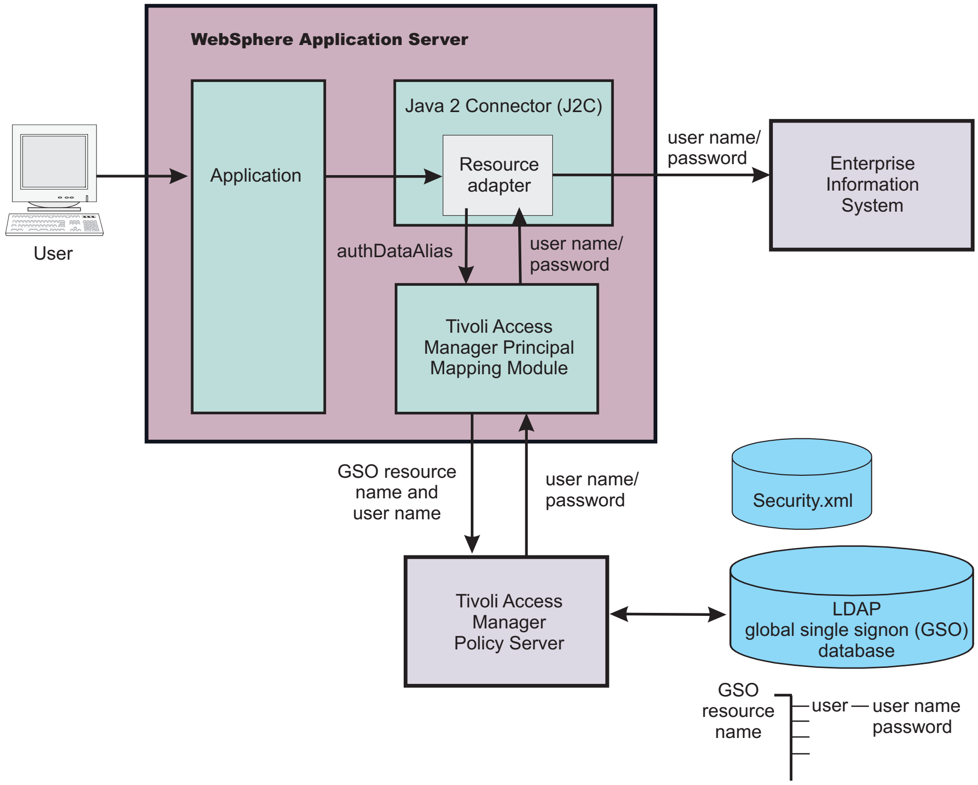
WebSphere Application Server provides a default principal mapping module that associates user credential information with EIS resources. The default mapping module is defined in the WebSphere Application Server administration console on the application login panel. To access the panel, click **Security** > **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


WebSphere Application Server
User
Application
Java 2 Connector (J2C)
Resource adapter
user name/ password
Enterprise Information System
authDataAlias
user name/ password
Tivoli Access Manager Principal Mapping Module
GSO resource name and user name
user name/ password
Security.xml
Tivoli Access Manager Policy Server
LDAP global single signon (GSO) database
GSO resource name
user
user name password

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 is 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 the **Connector Modules** link.

c. Click the **.rar** link.

d. Under Additional properties, click the **Resource Adapter** link.

   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 1531.

User registries

Information about users and groups reside in a user registry.

With 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 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.

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. Using 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. If the product processes in one node or cell need to communicate with other product processes in other nodes or cells using Lightweight Third Party Authentication (LTPA) (or Integrated Cryptographic Service Facility (ICSF) on the z/OS platform), all of the nodes and cells share the same user registry.

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 no registry is configured, the LocalOS 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. The following wsadmin command 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:

- Configure the local operating system user registry.
- Configure the LDAP user registry.
- Configure the custom user registries.

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.

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 provides implementations for the Windows local accounts registry and domain registry, as well as implementations for the Linux, Solaris, and AIX user accounts registries. Windows Active Directory is supported through the Lightweight Directory Access Protocol (LDAP) user registry implementation discussed later.

Note: For an Active Directory (domain controller), the three group scopes are Domain Local Group, Global Group, and Universal Group. For an Active Directory (Domain Controller), the two group types are Security and Distribution.

When a group is created, the default value is Global (and the default type is Security). With Windows NT domain registry support for Windows 2000 and 20003 domain controllers, WebSphere Application Server only supports Global groups that are the Security type. It is recommended that you use the Active Directory registry support (rather than an NT domain registry) if you use Windows 2000 and 2003 domain controllers because the Active Directory supports all group scopes and types. The Active Directory also supports a nested group that is not support by NT domain registry. The Active Directory is a centralized control registry.

WebSphere Application Server does not have to install the member of the domain because it can be installed on any machine on any platform. Note that the NT domain native call only returns the support group (with no error).

A Local OS user registry is not a centralized user registry like LDAP

Do not use a Local OS user registry in a WebSphere Application Server environment, where application servers are dispersed across more than one machine because each machine has its own user registry. Exceptions include a Windows domain registry, which is a centralized registry and Network Information Services (NIS), which is not supported by WebSphere Application Server.

As mentioned previously, the access IDs taken from the user registry are used during authorization checks. Because these IDs are typically unique identifiers, they vary from machine to machine, even if the exact users and passwords exist on each machine.

Web client certificate authentication is not currently supported when using the local operating system user registry. However, Java client certificate authentication does function with a local operating user registry. Java client certificate authentication maps the first attribute of the certificate domain name to the user ID in the user registry.

Even though Java client certificates function correctly, the following error displays in the `SystemOut.log` file:

```
CWSCJ0337E: The mapCertificate method is not supported
```

The error is intended for Web client certificates; however, it also displays for Java client certificates. Ignore this error for Java client certificates.

Using Windows operating system registries

When enabling security on Windows operating systems, if the local operating system (LocalOS) is selected as the registry, consider the following points:

Required privileges

The user that is running the WebSphere Application Server process requires enough operating system privilege to call the Windows systems application programming interface (API) for authenticating and obtaining user and group information from the Windows operating system. This user logs into the machine, or if running as a service, is the Log On As user. Depending on the machine and whether the machine is a stand-alone machine or a machine that is part of a domain or is the domain controller, the access requirements vary.

- For a stand-alone machine, the user:
  - Is a member of the administrative group.
  - Has the Act as part of the operating system privilege.
  - Has the Log on as a service privilege, if the server is run as a service.
- For a machine that is a member of a domain, only a domain user can start the server process and:
  - Is a member of the domain administrative groups in the domain controller.
  - Has the Act as part of the operating system privilege in the Domain security policy on the domain controller.
  - Has the Act as part of the operating system privilege in the Local security policy on the local machine.
  - Has the Log on as a service privilege on the local machine, if the server is run as a service.

    The user is a domain user and not a local user, which implies that when a machine is part of a domain, only a domain user can start the server.
- For a domain controller machine, the user:
  - Is a member of the domain administrative groups in the domain controller.
  - Has the Act as part of the operating system privilege in the Domain security policy on the domain controller.
  - Has the Log on as a service privilege on the domain controller, if the server is run as a service.

To give a user the Act as part of the operating system or Log on as a service on Windows 2000 systems:

1. Click **Start** > **Settings** > **Control Panel** > **Administrative Tools** > **Local Security Policy** > **Local Policies** > **User Rights Assignments** > **Act as part of the operating system** (or **Log on as a service**) .

   Note: If the machine is a stand-alone machine and not a member of a domain, you must add a machineName\userID, where the *userID* is the owner of the process, such as WebSphere Application Server. If you run WebSphere Application Server as a service, you can log on with `localsystem` as the service.

   If the machine is a member of a domain, add domainName\userID, where the *userID* is the owner of process (such as WebSphere Application Server). Start WebSphere Application Server as a service with login ID domainName\userID. If WebSphere Application Server is already in service, go to the service and right-click **IBM WebSphere Application Server > properties >Logon to change the logon ID and password** to restart WebSphere Application Server.
2. Add the user name by clicking **Add**.
3. Restart the machine.

   Windows 2000 domain controller users: For a Windows 2000 domain controller, replace **Local Security Policy** with **Domain Security Policy** in the previous step.

   Note: In all of the previous configurations, the server can be run as a service using `LocalSystem` for the Log On As entry. The `LocalSystem` entry has the required privileges and there is no need to give special privileges to any user. However, because the `LocalSystem` entry has special privileges, make sure that it is appropriate to use in your environment.

If the user running the server does not have the required privilege, you might see one of the following exception messages in the log files:

- `A required privilege is not held by the client.`
- `Access is denied.`

Domain and local registries

When WebSphere Application Server is started, the security run-time initialization process dynamically attempts to determine if the local machine is a member of a Windows domain. If the machine is part of a domain then by default both the local registry users or groups and the domain registry users or groups can be used for authentication and authorization purposes with the domain registry taking precedence. The list of users and groups that is presented during the security role mapping includes users and groups from both the local user registry and the domain user registry. The users and groups can be distinguished by the associated host names.

WebSphere Application Server does not support trusted domains.

If the machine is not a member of a Windows system domain, the user registry local to that machine is used.

Using both the domain 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.

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 localor 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.

Using UNIX system registries

When using UNIX system registries, the process ID that runs the WebSphere Application Server process needs the root authority to call the local operating system APIs for authentication and for obtaining user or group information.

Note: In UNIX systems, only the local machine registry is used. Network Information Service (NIS) (Yellow Pages) is not supported.

Using Linux and Solaris system registries

For WebSphere Application Server Local OS security registry to work on the Linux and Solaris platforms, a shadow password file must exist. The shadow password file is named shadow and is located in the /etc directory. If the shadow password file does not exist, an error occurs after enabling global security and configuring the user registry as Local OS.

To create the shadow file, run the **pwconv** command (with no parameters). This command creates an /etc/shadow file from the /etc/passwd file. After creating the shadow file, you can enable local operating system security successfully.

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 (`root` in UNIX, `Act as part of operating system` in Windows systems).

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:

For security purposes, the WebSphere Application Server provides and supports the implementation for Windows operating system registries, AIX, Solaris and multiple versions of Linux operating systems. The respective operating system APIs are called by the product processes (servers) for authenticating a user and other security-related tasks (for example, getting user or group information). Access to these APIs are restricted to users who have special privileges. These privileges depend on the operating system and are described below.

Before configuring the LocalOS registry you need to know the user name (ID) and password to use 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. 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 product startup. If authentication fails, the server does not come up. So it is important to choose an ID and password that do not expire or change often. If the product server user ID or password need to change in the registry, ensure that the changes are performed when all the product servers are up and running. After the changes are completed in the registry, use the following steps to change the ID and the password information. Save, stop, and restart all the servers so that the product can use the new ID or password. If any problem arises after starting the product because of authentication problems (that cannot be fixed), disable security before the server can start up. To avoid this step, make sure that the changes are validated in the Global Security panel. After the server is up, change the ID and password information and enable security.

When using the Windows operating system, consider the following issues:

- The server ID needs to be different from the Windows machine name where the product is installed. For example, if the Windows machine name is *vicky* and the security server ID is vicky, the Windows system fails when getting the information (group information, for example) for user *vicky*.
- WebSphere Application Server dynamically determines whether the machine is a member of a Windows system domain.
- WebSphere Application Server does not support Windows trusted domains.
- If a machine is a member of a Windows domain, both the domain user registry and the local user registry of the machine participate in authentication and security role mapping.
- The domain user registry takes precedence over the local user registry of the machine and can have undesirable implications if users with the same password exist in both user registries.
- The user that the product processes run under requires the `Administrative` and `Act as part of the operating system` privileges to call the Windows operating system APIs that authenticate or collect user and group information. The process needs special authority, which is given by these privileges. The user in this example might not be the same as the security server ID (the requirement for which is a valid user in the registry). This user logs into the machine (if using the command line to start the product process) or the Log On User setting in the services panel if the product processes have started using the services. If the machine is also part of a domain, this user is a part of the Domain Admin group in the domain to call the operating system APIs in the domain in addition to having the Act as part of operating system privilege in the local machine.

When using the UNIX operating systems (AIX and Solaris) and Linux, consider the following points:

- The user that the product processes run under requires the `root` privilege. This privilege is needed to call the UNIX operating system APIs to authenticate or to collect user and group information. The process needs special authority, which is given by the `root` privilege. This user might not be the same as the security server ID (the requirement is that it should be a valid user in the registry). This user logs into the machine and is running the product processes.
- On a UNIX operating system, the user that enables global security must have the `root` privilege if you use the local OS user registry. Otherwise, a failed validation error is displayed.
- When using the Linux operating system, you might need to have the password shadow file in your system.

The following steps are needed to perform this task initially when setting up security for the first time.

1. Click **Security > Global security**.
2. Under User registries, click **Local OS**,
3. Enter a valid user name in the Server user ID field.
4. Enter the user password in the Server user password field.
5. Click **OK**. Validation of the user and password does not happen 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 other steps and go to the Global Security panel. Make sure that LocalOS is the Active User Registry. If security was already enabled and you had changed either the user or the password information in this panel, make sure to go to the Global Security panel and click **OK** or **Apply** to validate your changes. If your changes are not validated, the server might not come up.

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**.

Server user ID:

Specifies a valid user ID in the local OS registry.

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. Access to native APIs is normally restricted to users having special privileges (for example, **root** in UNIX systems and **Act as part of operating system** in Windows systems). To use security in the application server, the process ID (not the security server ID) on which WebSphere Application Server runs requires enough privileges to call the system APIs. The special privilege means that the process running the WebSphere Application Server needs to be part of the **Administrators** group and have the **Act as part of operating system** privilege on Windows systems, and be **root**, or have root authority on UNIX systems.

Note: Windows If you are configuring Local OS security on Windows and you encounter the `A required privilege is not helped by the client` error message, you must follow the procedure documented to give the user those privileges. To set the privilege, click **Start > Settings > Control Panel > Administrative Tools > Local Security Policy > Local Policies > User Rights Assignments > Act as part of the operating system**.

When using a Windows system registry, this ID cannot match the name of the Windows machine. Windows systems treat the machine name bob as having an account similar to user bob.

Data type:	String
Units:	Alphanumeric characters

Server user password:

Specifies a valid user password that corresponds to a valid user ID in the local OS registry.

Data type	String

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 not supported.

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 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, Netscape, 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, iPlanet Directory Server 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 LDAP directory service 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, Netscape, 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 LDAP directory service documentation.

   For the IBM Directory Server, iPlanet Directory Server, 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

To use IBM Tivoli Directory Server (formerly IBM Directory Server), choose **IBM Tivoli Directory Server** as the directory type.

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.x. IBM Tivoli Directory Server, Version 5.2 includes WebSphere Application Server Express, Version 5.0.2, which the directory server uses for its administrative console. Install the Web Administration tool Version 5.2 and WebSphere Application Server Express, Version 5.0.2, which are both bundled with IBM Tivoli Directory Server, Version 5.2, on a different machine from WebSphere Application

Server, Version 6.x. You cannot use WebSphere Application Server, Version 6.x as the administrative console for IBM Tivoli Directory Server. If IBM Tivoli Directory Server, Version 5.2 and WebSphere Application Server, Version 6.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.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 5.0.2**.
- Install WebSphere Application Server, Version 6.x.
- When you install WebSphere Application Server, Version 6.x, change the port number for the application server.
- You might need to adjust the WebSphere Application Server environment variables on the version 6.x application server for *WAS_HOME* and *WAS_INSTALL_ROOT*. To change the variables using the administrative console, click **Environment > WebSphere Variables**.

Using a Lotus Domino Server as the LDAP server

If you choose the Lotus Domino LDAP server Version 6 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.

   For example, if the Active Directory administrator creates an account in the Users folder of the Active Directory Users and Computers Windows control panel and the DNS domain is `ibm.com`, the resulting DN has the following structure:

```
cn=<adminUsername>, cn=users, dc=ibm,
dc=com
```

2. Determine the short name and password of any account in the Microsoft Active Directory. 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`

      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 LDAP servers. For a list of supported LDAP servers, refer to the **Supported hardware, software and APIs** prerequisite Web site in the information center.

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 Directory Server, iPlanet Directory Server, 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, enable the Ignore case flag. This flag 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.

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 currently used in the static group membership search for all of the releases before WebSphere Application Server Version 5.

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 LDAP registry panel, select `SunONE` for the LDAP server.
2. Select the **Ignore case** 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 Directory Server: WebSphere Application Server Version 5.x or later supports all Lightweight Directory Access Protocol (LDAP) dynamic and nested groups when using IBM Directory Server 4.x and later. This function is enabled by default by taking advantage of a new feature in IBM Directory Server. IBM 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 Directory Server” for more information.

Configuring dynamic and nested group support for the IBM Directory Server:

When creating groups, ensure that nested and dynamic group memberships work correctly.

1. In the WebSphere Application Server security LDAP user registry configuration panel, select `IBM_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 methods″, in the information center, 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″ in the information center. 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. If you need to enter additional properties to initialize your implementation, click **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 1515 and the “groups.props file” on page 1515 article.

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, 2004
// 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 and groups.props files.

The contents of the `FileRegistrySample.java` file:

```
//
// 5639-D57, 5630-A36, 5630-A37, 5724-D18
// (C) COPYRIGHT International Business Machines Corp. 1997, 2003
// 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.
 *
 * 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.

```
//  @(#) 1.20 src/en/ae/rsec_result.xml, WEBSJAVA.INFO.DOCSRC,
//  ASVINFO1 10/17/02 16:43:01 [10/18/02 07:31:30]
//  5639-D57, 5630-A36, 5630-A37, 5724-D18
// (C) COPYRIGHT International Business Machines Corp. 1997, 2003
//  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, 2004
# 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, 2003
# 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 API helps enforce access control, based on the location of the code and the user. 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 for more information.

The following sections cover the JAAS implementation and programming model:
- Java Authentication and Authorization Service login configuration
- 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 1531 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 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 V1.0, 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 is problematic and causes undesirable effort. 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 API security document for applications to use. In the left navigation pane, click **Security > Global Security > JAAS Configuration > Application Login >WSLogin** and **ClientContainer**. The following three JAAS login configurations are available:

WSLogin

Defines a login configuration and a LoginModule implementation that applications can use in general.

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 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 J2EE Connector and the DefaultMappingModule module, refer to the J2EE security section.

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** in the navigation tree and expand the **Global Security** panel.
   b. 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** in the navigation tree and expand the **Global Security** panel.
   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. Click **JAAS Login Modules**.
   f. Click **New**.
   g. Specify the Module Classname. Specify 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. 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.

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 is problematic and causes undesirable effort. 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 administrative console (or by using the scripting functions) and stored in WebSphere configuration application programming interface (API). An application can define new JAAS login configuration in the Admin Console and the 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 define 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 API stores login module .jar files. 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.

Two JAAS login configurations are defined in the WebSphere configuration API security document for applications to use. To access the configurations, click **Security** > **JAAS Configuration** > **Application Login Config**: **WSLogin** and **ClientContainer**. 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)″ in the information center

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 artifacts 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 documentation 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:

- RMI_INBOUND
- WEB_INBOUND
- DEFAULT
- RMI_OUTBOUND
- SWAM
- wssecurity.IDAssertion
- wssecurity.signature
- wssecurity.PKCS7
- wssecurity.PkiPath
- wssecurity.UsernameToken
- wssecurity.X509BST
- LTPA
- LTPA_WEB

RMI_INBOUND, WEB_INBOUND, DEFAULT:

Processes 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 can 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 the credential mapping services, 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 protocol, click **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 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.

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 used by the Web container such as servlets and JavaServer pages (JSP) files.

This login configuration is used by WebSphere Application Server Version 5.1. This login configuration was introduced in version 5.1 and is no longer used in version 5.1.1.

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, click **Security > Global security**. Under Authentication, click **JAAS configuration > Application logins** or **System logins** > *alias_name*. 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
- Kerbv5: Kerberos Version 5-based authentication mechanism

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 synchronized 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.

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 in the information center.

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 authenticated 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** in the navigation tree and select **Global Security**, and then click **JAAS Configuration** > **J2C Authentication Data**. The **J2C Authentication Data Entries** panel is displayed.
   b. 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** in the navigation tree and select **Global Security**, and then click **JAAS Configuration** > **J2C Authentication Data**. The **J2C Authentication Data Entries** panel is displayed.
b. Click **New**.
c. Enter a unique alias, a valid user ID, a valid password, and a short description (optional).
d. Click **OK** or **Apply**. No validation for the user ID and password is required.
e. 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, you can click either **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

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.″

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.″

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 the information center. 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″
 //  in the information center.

 _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″
 //  in the information center.
   // 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.
  // 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"
 //  in the information center.

 // 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.

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 1547.
  2. Use the code sample in "Example: Using the WSLogin configuration to create a basic authentication subject" on page 1547 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″ in the information center.

        The "Example: Sample login configuration for RMI_OUTBOUND" on page 1548 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"
  // in the information center.
 }

 public boolean login() throws LoginException
 {
     // (For more information on what to do during login, see
     // "Custom login module development for a system login configuration"
  // in the information center.

  // 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"
 // in the information center.
```

```
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 1557 and “Implementing a custom PropagationToken” on page 1563.

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 1448. For information on how to use and implement this token type, see “Default SingleSignonToken” on page 1585 and “Implementing a custom SingleSignonToken” on page 1586.

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 1557 and “Implementing a custom PropagationToken” on page 1563.

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 1597 and “Implementing a custom AuthenticationToken” on page 1598.

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:

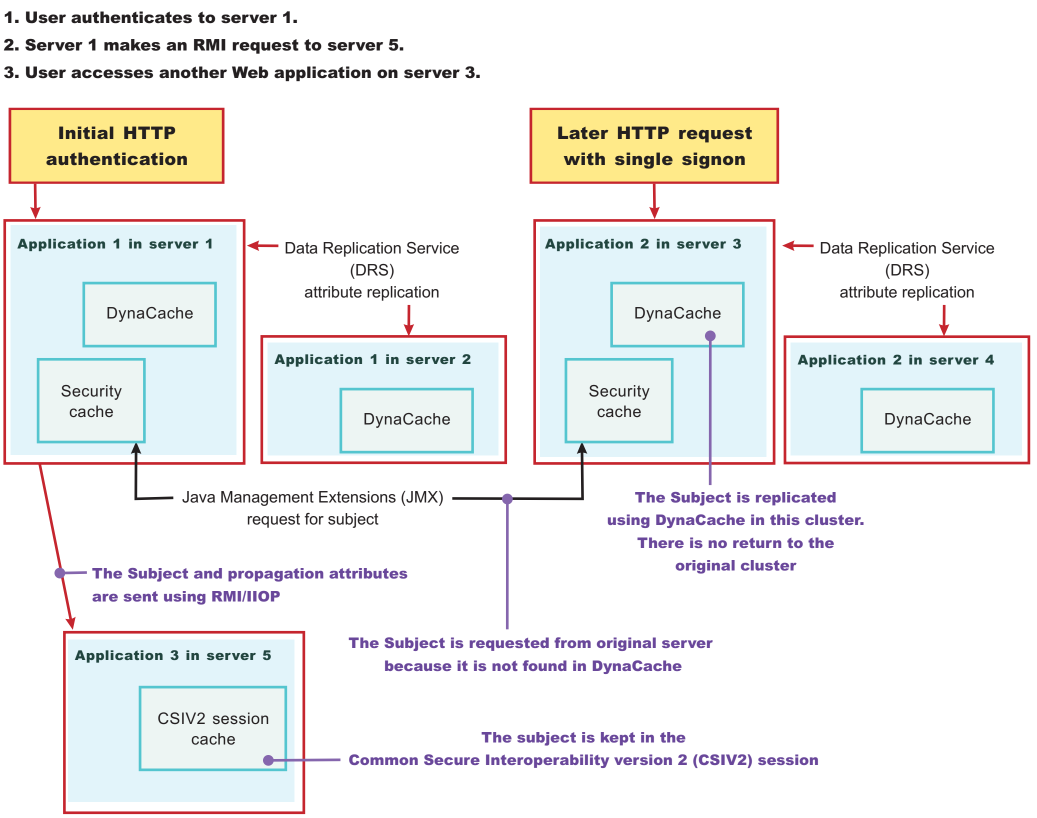
1. Click **Security > Global security**.
2. Under Authentication, click **Authentication Mechanisms > LTPA**.
3. Under Additional properties, click **Single signon (SSO)**

For more information, see “Enabling security attribute propagation” on page 1555.

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


1. User authenticates to server 1.
2. Server 1 makes an RMI request to server 5.
3. User accesses another Web application on server 3.
Initial HTTP authentication
Later HTTP request with single signon
Application 1 in server 1
DynaCache
Security cache
Data Replication Service (DRS) attribute replication
Application 1 in server 2
DynaCache
Application 2 in server 3
DynaCache
Security cache
Data Replication Service (DRS) attribute replication
Application 2 in server 4
DynaCache
Java Management Extensions (JMX) request for subject
The Subject is replicated using DynaCache in this cluster. There is no return to the original cluster
The Subject and propagation attributes are sent using RMI/IIOP
Application 3 in server 5
CSIV2 session cache
The Subject is requested from original server because it is not found in DynaCache
The subject is kept in the Common Secure Interoperability version 2 (CSIV2) session

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.

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 causes frequent downstream calls. If you change the token prior to making many downstream calls, but 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*/*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 host 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 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 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 1564
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 1602. 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 1570 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″ in the information center.

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 1563.

```
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"
  // in the information center.
 }

 public boolean login() throws LoginException
 {
  // (For more information on what to do during login, see
     // "Custom login module development for a system login configuration"
  // in the information center.


  // 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″
  // in the information center.
  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 1557. 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″ in the information center.

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″
  // in the information center.

  // 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″
  // in the information center.

  // 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"
  // in the information center.

 }

 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() 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 1602. After the object is instantiated in the login module, you can 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 1602. 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 1582 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″ in the information center.

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 1576.

```
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″
     // in the information center.

  _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″
     // in the information center.

  // 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″
     // in the information center.

  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 1586 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 1587

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 1592 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 token 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 1594 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″ in the information center.

After completing these steps, you have implemented a custom AuthorizationToken.

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 1586.

```
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"
    // in the information center.

  _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"
    // in the information center.

     // 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"
    // in the information center.

  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 1586.

```
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″
     // in the information center.

  _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″
     in the information center.

     // 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"
     // in the information center.

  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 1598 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() 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.AuthenticationToken implementation″ in the information center.

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 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 1602. 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 1600 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″ in the information center.

After completing these steps, you have implemented a custom AuthenticationToken.

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″
   // in the information center.

  _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″
   // in the information center.

  // 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"
  // in the information center.

 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, deserializes the object, and adds the object back into the Subject downstream. However, there are some requirements for this process to occur. See ″Security: Resources for learning″ in the information center.

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″
  // in the information center.)
 }

 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″
  // in the information center.)

     // 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"
 // in the information center.)

 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″ 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 1573.

   PropagationToken
   Add attributes that are specific to an invocation. For more information, see “Default PropagationToken” on page 1557.

   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″ in the information center.
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 1555.
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″ in the information center.

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 adminConsole application, MBeans, and other components such as naming and security. For more information on administrative security, see ″Role-based authorization″ in the information center.

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 1619 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 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 include the storing of the policy information in their repository during application deployment. The providers then use this information to make authorization access 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 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 (for example, by using the Trust Association Interface feature or by plugging login modules into the Application Server), that information is available in the Subject. The provider can then make use of the information added in the Subject to make the access decision.

The security attribute propagation has more information on how to add information to the Subject. See “Enabling security attribute propagation” on page 1555 for more information.

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 any security policies are modified in the Web modules, the application is stopped and then restarted when using the default authorization engine. When using the Java 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 1617 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 is **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 1612.

Initialization of the JACC provider

If a Java 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 1619 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 Contract for Containers (JACC) is a new feature in WebSphere Application Server Version 6. 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 only contains 6.0 nodes. 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 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 has 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 that 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 1619 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 changed as part of the update. If the provider supports the RoleConfiguration interfaces, the entire authorization table for that application is propagated to the provider.

If for some reason, 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). However, the application is not propagated to its respective nodes until the sychronization command is issued and completed. Also, in the ND setup 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 1617 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.

Enabling an external JACC provider

The Java 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** from the left navigation menu.
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. Unless you want 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, use the **Custom properties** link to 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 under **Security > Global Security > Authorization Providers** and then click **OK**.
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 1474.
8. 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
   ```
9. 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.

For more information on JACC support in WebSphere Application Server, refer to the information center documentation.

Use these settings when you have set up an external security provider to work with WebSphere Application Server that can support Java 2 Platform, Enterprise Edition (J2EE) authorization based on the JACC specification. The setup 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

It is possible that you have applications installed prior to enabling the 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 so that the JACC provider is updated with this information. However, since reinstallation might not be an option in some cases, the wsadmin 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 1619 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 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 is 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 1608
- "JACC providers" on page 1607
- "Tivoli Access Manager integration as the JACC provider" on page 1622
-

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=inm`, `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 1613 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 for every process start. 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 is 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. The information is stored in the security policy database in the Tivoli Access Manager policy server.

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 1619 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. The setup can be done either before (using wsadmin) or during the time of WebSphere Application Server configuration.

For more information about how to configure the Tivoli Access Manager client, see “Tivoli Access Manager JACC provider configuration” on page 1626.

Authorization table support

Tivoli Access Manager uses the RoleConfiguration interface to ensure that the authorization table information is passed to the Tivoli Access Manager provider when the application is installed or deployed. When an application is deployed or edited, the set of users and groups for the user or group-to-role mapping are obtained from the Tivoli Access Manager server, which shares the same Lightweight Directory Access Protocol (LDAP) server as WebSphere Application Server. This sharing is accomplished by plugging into the application management users or groups-to-role administrative console panels. The management APIs are called to obtain users and groups rather than relying on the WebSphere Application Server-configured LDAP registry.

Access check

When WebSphere Application Server is configured to use the JACC provider for Tivoli Access Manager , it passes the information to Tivoli Access Manager to make the access decision. The Tivoli Access Manager policy implementation queries the local replica of the access control list (ACL) database for the access decision.

Authentication using the PDLoginModule module

The custom login module in WebSphere Application Server can do the authentication. This login module is plugged in before the WebSphere Application Server-provided login modules. The custom login modules can provide information that can be stored in the Subject. If the required information is stored, no additional registry calls are made to obtain that information.

As part of the JACC integration, the Tivoli Access Manager-provided PDLoginModule module is also used to plug into WebSphere Application Server for both Lightweight Third Party Authentication (LTPA) and Simple WebSphere Authentication Mechanism (SWAM) authentication. The PDLoginModule module is modified to authenticate with the user ID or password. The module is also used to fill in the required attributes in the Subject so that no registry calls are made by the login modules in WebSphere Application Server. The information that is placed in the Subject is available for the Tivoli Access Manager policy object to use for access checking.

Tivoli Access Manager security for WebSphere Application Server

WebSphere Application Server Version 6.0 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 imbedded 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 Tivoli Access Manager Web Portal Manager GUI
- No requirements for coding or deployment changes to applications
- Easy management of users, groups, and roles using the WebSphere Application Server administrative console

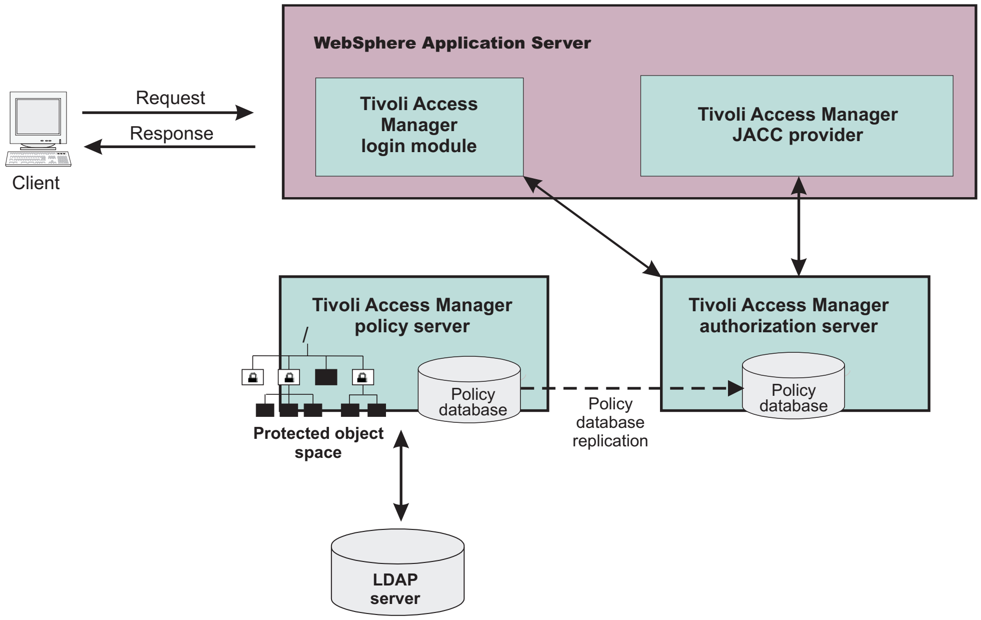
WebSphere Application Server Version 6.0 supports the Java Authorization Contract for Containers (JACC) specification. JACC details the contract requirements for J2EE containers and authorization providers. With this detail, authorization providers can perform the access decisions for resources in J2EE 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 is JACC-compliant and is used to:

- Add security policy information when applications are deployed
- Authenticate users
- 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


WebSphere Application Server
Tivoli Access Manager login module
Tivoli Access Manager JACC provider
Request
Response
Client
Tivoli Access Manager policy server
/
Policy database
Protected object space
Policy database replication
Tivoli Access Manager authorization server
Policy database
LDAP server

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.

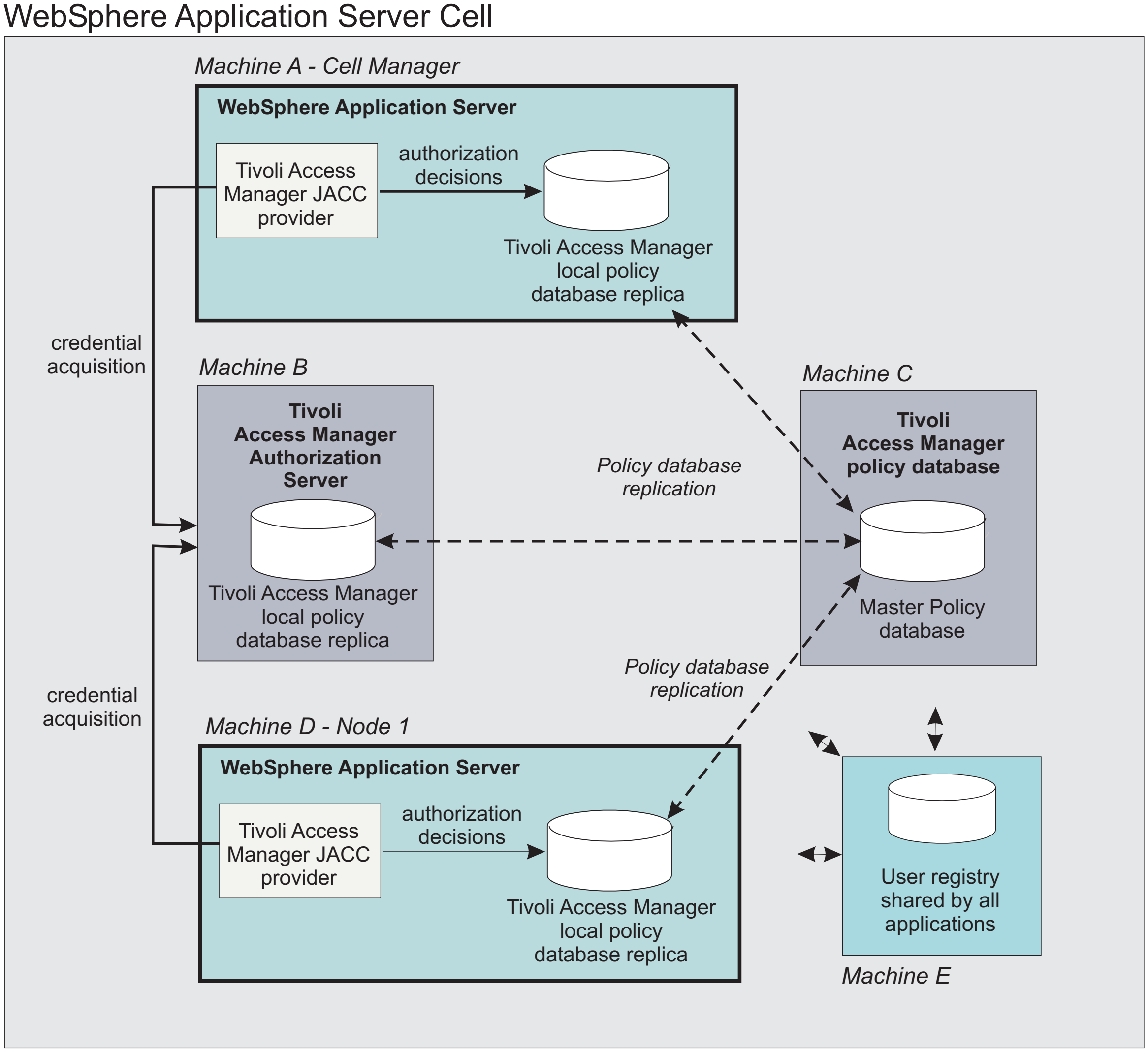
At its core, Tivoli Access Manager provides an authentication and authorization framework. You can learn more about Tivoli Access Manager, including information that is necessary to make deployment decisions, by reviewing the product documentation. Start with the following guides, available at http://publib.boulder.ibm.com/tividd/td/tdprodlist.html:

- *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.


WebSphere Application Server Cell
Machine A - Cell Manager
WebSphere Application Server
Tivoli Access Manager JACC provider
authorization decisions
Tivoli Access Manager local policy database replica
credential acquisition
Machine B
Tivoli Access Manager Authorization Server
Tivoli Access Manager local policy database replica
Policy database replication
Machine C
Tivoli Access Manager policy database
Master Policy database
Policy database replication
credential acquisition
Machine D - Node 1
WebSphere Application Server
Tivoli Access Manager JACC provider
authorization decisions
Tivoli Access Manager local policy database replica
User registry shared by all applications
Machine E

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 that are installed as part of the Tivoli Access Manager installation. Having policy database replicas on each participating WebSphere Application Server optimizes performance when making authorization decisions and provides failover capability.

The authorization server can also be installed on the same system as WebSphere Application Server, although 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 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/`*`profileName`*`/cells/`*`cellName`*.

```
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 1629
- 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 `etc/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.
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 1626. 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 1632.

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 1626. 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
```

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

Field	Value
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 JACC provider** option.
4. The JACC provider settings for Tivoli Access Manager are displayed. Click **OK**.
5. Save the settings by clicking **Save** at the top of the page; click the **Save** button.
6. Log out of the WebSphere Application Server administrative console.
7. 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.

	Methods		
Roles	**getBalance**	**deposit**	**closeAccount**
Teller	granted	granted	
Cashier	granted		
Supervisor			granted

In the example of the banking application that is conceptualized in the previous table, three roles are defined: teller, cashier, and supervisor. Permission to perform the getBalance, deposit, and closeAccount application methods are mapped to these roles. From the example, you can see that users assigned the role, Supervisor, can run the closeAccount method, whereas the other two roles are unable to run this method.

The term, principal, within WebSphere Application Sever security refers to a person or a process that performs activities. Groups are logical collections of principals that are configured in WebSphere Application Server to promote the ease of applying security. Roles can be mapped to principals, groups, or both. The entry invoked in the following table indicates that the principal or group can invoke any methods that are granted to that role.

	Roles		
Principal/Group	Teller	Cashier	Supervisor
TellerGroup	Invoke		
CashierGroup		Invoke	
SupervisorGroup			
Frank - a principal who is not a member of any of the previous groups		Invoke	Invoke

In the previous example, the principal Frank, can invoke the getBalance and the closeAccount methods, but cannot invoke the deposit method because this method is not granted either the Cashier or the Supervisor role.

At the time of application deployment, the Java Authorization Contract for Container (JACC) provider of Tivoli Access Manager populates the Tivoli Access Manager-protected object space with any security policy information that is contained in the application deployment descriptor. This security information is used to determine access whenever the WebSphere 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 JACC provider messages are logged to the WebSphere Application Server file, `SystemOut.log` . Trace logging is sent to the WebSphere Application Server file, `trace.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** check box.
   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 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 1626.
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 1626.
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 1474.
4. Enable the Tivoli Access Manager JACC provider. For more information, see “Enabling the JACC provider for Tivoli Access Manager” on page 1632.

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 1642.

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 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-authzn.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-authzn.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-authzn.xml`. Use the parameter descriptions listed in The Tivoli Access Manager migrateEAR5 utility. For example:

```
migrateEAR
\ -j "c:\Program Files\IBM\WebSphere\AppServer\profiles\profileName\config\
     cells\cellName\admin-authz.xml"
\ -a sec_master
\ -p password
\ -w wsadmin
\ -d o=ibm,c=us
\ -c file:/"c:\Program Files\IBM\WebSphere\AppServer\java\jre\PdPerm.properties"
```

   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-authzn.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:

Solaris, Linux and HP-UX -
`file:/opt/IBM/WebSphere/AppServer/java/jre/PdPerm.properties`AIX –
`file:/usr/IBM/WebSphere/AppServer/java/jre/PdPerm.properties`
Windows –
`file:/"c:\Program Files\IBM\WebSphere\AppServer\java\jre\PdPerm.properties"`

-d user_registry_domain_suffix
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:

Solaris, Linux and HP-UX -
```
file:/opt/IBM/WebSphere/AppServer/profiles/profileName/config/cells/
     cellName/admin-authz.xml
```
AIX –
```
file:/usr/IBM/WebSphere/AppServer/profiles/profileName/config/cells/
     cellName/admin-authz.xml
```
Windows –
```
"c:\Program Files\IBM\WebSphere\AppServer\profiles\profileName\config\
     cells\cellName\admin-authz.xml"
```

-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 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 a Tivoli Access Manager administrative user for WebSphere Application Server. 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 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 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 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."
```

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: Secure Authentication Service (SAS) and Common Secure Interoperability Version 2 (CSIv2). SAS is the authentication protocol used by all previous releases of WebSphere Application Server and is maintained for backwards compatibility. The Object Management Group (OMG) has defined the authentication protocol called CSIv2 so that vendors can interoperate securely. CSIv2 is implemented in WebSphere Application Server with more features than SAS and 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.

Choose a protocol by specifying the `com.ibm.CSI.protocol` property on the client side and configuring through the administrative console on the server side. More details are included in the SAS and CSIv2 properties articles.

Common Secure Interoperability Specification, Version 2

The Common Secure Interoperability Specification, Version 2 (CSIv2) defines the Security Attribute Service (SAS) that enables interoperable authentication, delegation and privileges. The CSIv2 SAS and SAS protocols are entirely different. The CSIv2 SAS is a subcomponent of CSIv2 that supports SSL and interoperability with the EJB Specification, Version 2.1.

Security Attribute Service

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.

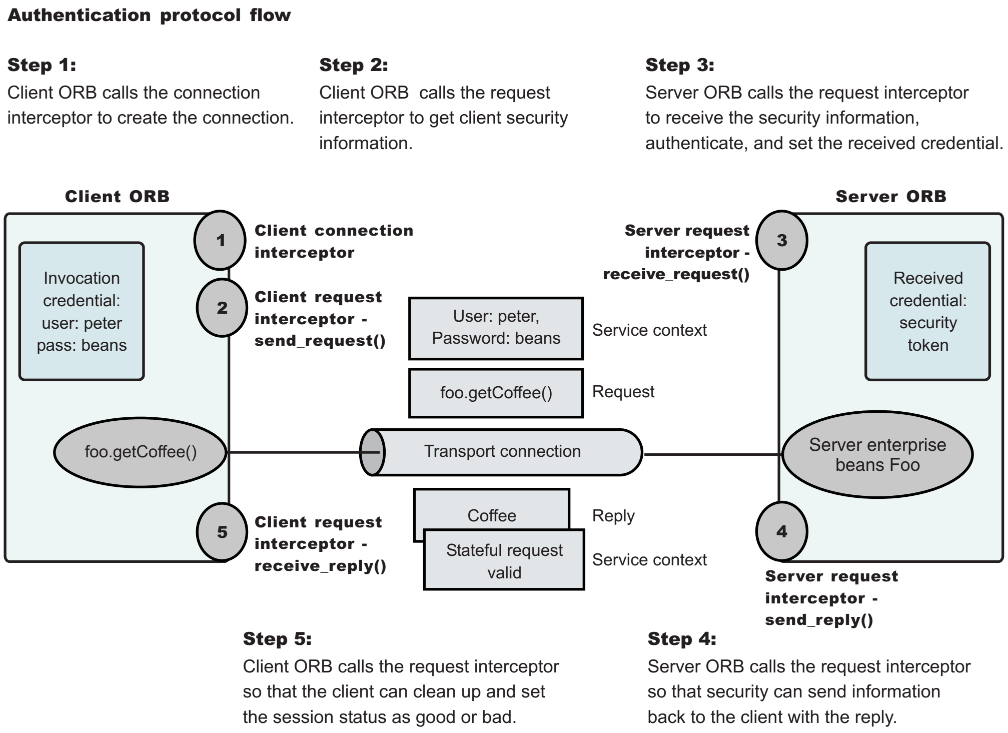
If a client certificate chain is found, the distinguished name (DN) is extracted from the certificate and is used to map to an identity in the user registry. If the user registry is Lightweight Directory Access Protocol (LDAP), the search filters defined in the LDAP registry configuration determine how the certificate maps to an entry in the registry. If the user registry is local OS, the first attribute of the distinguished name (DN) maps to the user ID of the registry. This attribute is typically the common name.

If the certificate does not map, no credential is created and the request is rejected. When 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.

Specifying a stateful client is done through the property com.ibm.CSI.performStateful (true/false). Specifying a stateful server is done through the administrative console configuration.


Authentication protocol flow
Step 1:
Client ORB calls the connection interceptor to create the connection.
Step 2:
Client ORB calls the request interceptor to get client security information.
Step 3:
Server ORB calls the request interceptor to receive the security information, authenticate, and set the received credential.
Client ORB
Server ORB
1
Client connection interceptor
2
Client request interceptor - send_request()
Invocation credential: user: peter pass: beans
User: peter, Password: beans
Service context
foo.getCoffee()
Request
foo.getCoffee()
Transport connection
3
Server request interceptor - receive_request()
Received credential: security token
Server enterprise beans Foo
5
Client request interceptor - receive_reply()
Coffee
Reply
Stateful request valid
Service context
4
Server request interceptor - send_reply()
Step 5:
Client ORB calls the request interceptor so that the client can clean up and set the session status as good or bad.
Step 4:
Server ORB calls the request interceptor so that security can send information back to the client with the reply.

. *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.

- SSL Client Certificate authentication.

  An additional way to authenticate a client to a server using SSL client authentication.
- Message Layer Authentication.

  Authenticates credential information and sends that information across the network so that a receiving server can interpret it.
- 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.
- 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 authorization checks the credential to determine whether access is permitted, it also sets the *invocation* credential so that if the Enterprise JavaBeans (EJB) method calls another EJB method that is located on other servers, the invocation credential can be the identity used to invoke the downstream method. Depending on the RunAs mode for the enterprise beans, the invocation credential is set as the originating client identity, the server identity, or a specified different identity. Regardless of the identity that is set, when identity assertion is enabled, it is the invocation credential that is asserted to the downstream server.

The invocation credential identity is sent to the downstream server in an identity token. In addition, the sending server identity, including the password or token, is sent in the client authentication token when basic authentication is enabled. The sending server identity is sent through a Secure Sockets Layer (SSL) client certification authentication when client certificate authentication is enabled. Basic authentication takes precedence over client certificate authentication.

Both 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.

Evaluation of the identity token consists of the following four identity formats that exist in an identity token:

- Principal name
- Distinguished name
- Certificate chain
- Anonymous identity

The product servers that receive authentication information typically support all four identity types. The sending server decides which one is chosen, based on how the original client authenticated. The existing type depends on how the client originally authenticates to the sending server. For example, if the client uses Secure Sockets Layer (SSL) client authentication to authenticate to the sending server, then the identity token sent to the downstream server contains the certificate chain. With this information, the receiving server can perform its own certificate chain mapping and interoperability is increased with other vendors and platforms.

After the identity format is understood and parsed, the identity maps to a credential. For an ITTPrincipal identity token, this identity maps one-to-one with the user ID fields.

For an ITTDistinguishedName identity token, the mapping depends on the user registry. For Lightweight Directory Access Protocol (LDAP), the configured search filter determines how the mapping occurs. For LocalOS, the first attribute of the distinguished name (DN), which is typically the same as the common name, maps to the user ID of the registry. For an ITTCertChain identity token, see the Map certificates to users section for details on how this action is performed for the LDAP user registry. For LocalOS, the first attribute of the DN in the certificate is used to map to the user ID in the registry.

Some user registry methods are called to gather additional credential information that is used by authorization. In a stateful server, this action completes once for the sending server and the receiving server pair where the identity tokens are the same. Subsequent requests are made through a session ID.

Identity assertion is only available using the Common Secure Interoperability Version 2 (CSIv2) protocol.

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 (whether the token is a user ID and password token, that is, Generic Security Services Username Password (GSSUP), or a mechanism-specific format token, Lightweight Third Party Authentication (LTPA), for example on non-z/OS platforms), 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.

However, a servlet can use either basic authentication (GSSUP) or the authentication mechanism of the server (LTPA) to send security information in the message layer. Use LTPA by authenticating or by mapping the basic authentication credentials to the security mechanism of the server.

The security token that is contained in a token-based credential is authentication mechanism-specific. The way that the token is interpreted is only known by the authentication mechanism. Therefore, each authentication mechanism has an object ID (OID) representing it. The OID and the client token are sent to the server, so that the server knows which mechanism to use when reading and validating the token. The following list contains the OIDs for each mechanism:

```
BasicAuth  (GSSUP):   oid:2.23.130.1.1.1
LTPA:       oid:1.3.18.0.2.30.2
SWAM:        No  OID  because  it  is  not  forwardable
```

On the server, the authentication mechanisms can interpret the token and create a credential, or they can authenticate basic authentication data from the client, and create a credential. Either way, the created credential is the *received* credential that the authorization check uses to determine if the user has access to invoke the method. You can specify the authentication mechanism by using the following property on the client side:

- com.ibm.CORBA.authenticationTarget

Basic authentication is currently the only valid value. You can configure the server through the administrative console.

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.

Configuring authentication retries

Situations occur where you want a prompt to display again if you entered your user ID and password incorrectly or you want a method to retry when a particular error occurs back at the client. If you can correct the error by information at the client side, the system automatically performs a retry without the client seeing the failure, if the system is configured appropriately.

Some of these errors include:

- Entering a user ID and password that are not valid
- Having an expired credential on the server
- Failing to find the stateful session on the server

By default, authentication retries are enabled and perform three retries before returning the error to the client. Use the com.ibm.CORBA.authenticationRetryEnabled property (`True` or `False`) to enable or disable authentication retries. Use the com.ibm.CORBA.authenticationRetryCount property to specify the number of retry attempts.

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 SSL keystore 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.

- The ORB proceeds to call the client request interceptor, which might be busy.

  If basic authentication is also configured, for example, then the user might be prompted for a user ID and password. Because this action is not necessary, disable this option in the configuration if the SSL certificate is the identity against which to invoke the method. If no message layer security exists, then no security context is created and associated with the request.
- 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.

  See the "Map certificates to users" on page 1722 article, for details on how this mapping is performed for the Lightweight Directory Access Protocol (LDAP) user registry. For local OS, the first attribute of the distinguished name (DN) in the certificate is used to map to the user ID in the registry.

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 the keystore file on each client system.

To enable SSL client certificate authentication on the client side, you must enable the properties, such as SSL. 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.

SSL client certificate authentication from a Java client is only available using the Common Secure Interoperability Version 2 (CSIv2) protocol.

Supported authentication protocols: Two authentication protocols are supported. Secure Authentication Service (SAS) is the authentication protocol used by all previous releases of the WebSphere Application Server product. Common Secure Interoperability Version 2 (CSIv2), which is considered the strategic protocol, is implemented in WebSphere Application Server, Version 5 and later.

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 and CSIv2. If a client supports both protocols, it reads tagged components for both CSIv2 and SAS. If the client and the server support both protocols, CSIv2 is used. However, if the server supports SAS (for example, it is a previous WebSphere Application Server release) and the client supports both protocols, the client chooses SAS for this request.

Choose a protocol using the com.ibm.CSI.protocol property on the client side and configure this protocol through the administrative console on the server side.

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.

   The Java client utilizes the `sas.client.props` file to configure outbound security. Pure clients need to configure outbound security only.

   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, authentication by a user ID and password is performed.

   Both Java client certificate authentication and identity assertion are disabled by default. If you want this type of basic 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 the `sas.client.props` file, where properties are modified.

   Configuring servers is always done from the administrative console, 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 a valid Web address with the protocol type, `file`.

```
-Dcom.ibm.CORBA.ConfigURL=file:/C:/WebSphere/AppServer/properties/sas.client.props
```

When this file is processed by the Object Request Broker (ORB), security can be enabled between the Java client and the target server.

If any syntax problems exist with the ConfigURL property and the `sas.client.props` file is not found, the Java client proceeds to connect insecurely. Errors display indicating the failure to read the ConfigURL property. Typically the problem is related to having two slashes after `file`, which is not valid.

Use the following properties to configure the SAS and CSIv2 authentication protocols:

- "CSIv2 authentication protocol client settings"
- "Security Authentication Service authentication protocol client settings" on page 1660

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.performStateful:

Used to determine if the CSIv2 protocol maintains stateful sessions between a client and server after the initial secure association (authentication between a particular client and server).

For performance reasons, it is beneficial to enable this property. Considerations for disabling this property include troubleshooting an authentication protocol session-related problem.

Data type:	Boolean
Default:	True
Range:	True or False

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) or Kerberos, 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.performMessageConfidentialitySupported:

Use to determine if 128-bit ciphers are supported to make SSL connections.

If a target server does not support 128-bit ciphers, you can make a connection at a lower encryption strength. This property is only valid when SSL is enabled.

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.performMessageIntegritySupported:

Use to determine if 40-bit ciphers are supported to make SSL connections.

If a target server does not support 40-bit ciphers, you can make a connection using only digital-signing ciphers. This property is only valid when SSL is enabled. This property is ignored if the associated com.ibm.CSI.performMessageIntegrityRequired property is enabled.

Data type:	Boolean
Default:	True
Range:	True or False

com.ibm.CSI.performMessageIntegrityRequired:

Use to determine if 40-bit ciphers must be used to make SSL connections.

If a target server does not support 40-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.performMessageIntegritySupported property is ignored.

Data type:	Boolean
Default:	True
Range:	True or False

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.

Security Authentication Service authentication protocol client settings:

In addition to those properties which are valid for both Security Authentication Service (SAS) and Common Secure Interoperability Version 2 (CSIv2), this article documents properties which are valid only for the SAS authentication protocol.

com.ibm.CORBA.standardPerformQOPModels:

Specifies the strength of the ciphers when making an Secure Sockets Layer (SSL) connection.

Data type:	String
Default:	High
Range	Low, Medium, High

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 is authenticating on this server. Trust of the upstream server is established either using Secure Sockets Layer (SSL) client certificate authentication or basic authentication. You must select one of the two layers of authentication in both inbound and outbound authentication when you choose identity assertion.

     The server ID is sent in the client authentication token with the identity token. The server ID is checked against the trusted server ID list. If the server ID is on the trusted server list, the server ID is authenticated. If the server ID is valid, the identity token is put into a credential and used for authorization of the request.
   - User ID and password (message layer).

     This type of authentication is the most typical. The user ID and password or authenticated token is sent from a pure client or from an upstream server. However, the upstream server cannot be a z/OS server because z/OS does not support a user ID or password from a server acting as a client. When a user ID and password are received at the server, they are authenticated with the user registry.

     Usually, a token is sent from an upstream server and a user ID and password are sent from a client, including a servlet. When a token is received at the server level, the token is validated to determine whether tampering has occurred or whether it is expired.
   - 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.

     In some cases, if the client authenticated with a token and a Lightweight Directory Access Protocol (LDAP) server is the user registry, then a distinguished name (DN) is sent.
6. Configure a trusted server list. When identity assertion is selected for inbound requests, insert a pipe-separated (|) list of server administrator IDs to which this server can support identity token submission. For backwards compatibility, you can still use a comma-delimited list. However, if the server ID is a distinguished name (DN), then you must use a pipe-delimited (|) list because a comma delimiter does not work. If you choose to support any server sending an identity token, you can enter an asterisk (*) in this field. This action is called *presumed trust*. In this case, use SSL client certificate authentication between servers to establish the trust.
7. Configure session management. You can choose either *stateful* or *stateless* security. Performance is optimum when choosing stateful sessions. The first method request between a client and server is authenticated. All subsequent requests (or until the credential token expires) reuse the session information, including the credential. A client sends a context ID for subsequent requests. The context ID is scoped to the connection for uniqueness.

When you finish configuring this panel, you have configured most of the information that a client 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.

For an enterprise bean 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 three layers of authentication that you can use simultaneously:

- **Transport layer**. The transport layer, which is the lowest layer, might contain a Secure Sockets Layer (SSL) client certificate as the identity.
- **Message layer**. The message layer might contain a user ID and password or an authenticated token with an expiration.
- **Attribute layer**. The attribute layer might contain an identity token, which is an identity from an upstream server that already is authenticated. The identity layer has the highest priority, followed by the message layer, and then the transport layer. If a client sends all three, only the identity layer is used. The only way to use the SSL client certificate as the identity is if it is the only information that is presented during the request. The client picks up the interoperable object reference (IOR) from the namespace and reads the values from the tagged component to determine what the server needs for security.

Basic authentication:

Specifies that basic authentication occurs over the message layer.

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 has the signer certificate of each client to which it is connected. If the client uses a certificate authority (CA) to create its personal certificate, you only need the CA root certificate in the server signer section of the SSL trust file.

When the certificate is authenticated to a Lightweight Directory Access Protocol (LDAP) user registry, the distinguished name (DN) is mapped based on the filter that is specified when configuring LDAP. When the certificate is authenticated to a LocalOS user registry, the first attribute of the distinguished name (DN) in the certificate, which is typically the common name, is mapped to the user ID in the registry.

The identity from client certificates is used only if no other layer of authentication is presented to the server.

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. 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`.

When authenticating to a LocalOS user registry, 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 pipe-separated (|) list of trusted server IDs, which are trusted to perform identity assertion to this server. For example, `serverid1|serverid2|serverid3`. WebSphere Application Server supports the comma (,) character as the list delimiter for backwards compatibility. WebSphere Application Server checks the comma character when the pipe character (|) fails to find a valid trusted server ID.

Use this list to decide whether a server is trusted. Even if the server is on the list, the sending server must still authenticate with the receiving server to accept the identity token of the sending server.

Data type	String

Stateful sessions:

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.

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 to go 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 submits 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 on the socket. The same scenario is true for other mechanisms of authentication. Read the ″Identity Assertion″ article in the information center for details.
2. Select **User ID** and **Password** (message layer). This type of authentication is the most typical. The user ID and password (if BasicAuth credential) or authenticated token (if authenticated credential) are sent outbound to the downstream server if the downstream server supports message layer authentication in the inbound authentication panel. Refer to the Message Layer Authentication article for more information.
3. 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.

When you finish configuring this panel, you configured the information that this server uses to make decisions about the type of authentication to perform with downstream servers. If the downstream server is configured not to support the outbound configuration of the server, the following exception likely occurs:

```
Exception received: org.omg.CORBA.INITIALIZE:
CWWSA1477W: SECURITY CLIENT/SERVER CONFIG MISMATCH:  The client security
configuration (sas.client.props or outbound settings in GUI) does not
```

```
support the server security configuration for the following reasons:
ERROR 1: CWWSA0607E: The client requires SSL Confidentiality but the server
does not support it.
ERROR 2: CWWSA0610E: The server requires SSL Integrity but the client does
not support it.
ERROR 3: CWWSA0612E: The client requires client (e.g., userid/password or token),
but the server does not support it.
minor code: 0  completed: No
        at com.ibm.ISecurityLocalObjectBaseL13Impl.SecurityConnectionInterceptor.
getConnectionKey(SecurityConnectionInterceptor.java:1770)
        at com.ibm.ws.orbimpl.transport.WSTransport.getConnection(Unknown Source)
        at com.ibm.rmi.iiop.TransportManager.get(TransportManager.java:79)
        at com.ibm.rmi.iiop.GIOPImpl.locate(GIOPImpl.java:167)
        at com.ibm.CORBA.iiop.ClientDelegate._createRequest(ClientDelegate.java:2088)
        at com.ibm.CORBA.iiop.ClientDelegate.createRequest(ClientDelegate.java:1264)
        at com.ibm.CORBA.iiop.ClientDelegate.createRequest(ClientDelegate.java:1177)
        at com.ibm.CORBA.iiop.ClientDelegate.request(ClientDelegate.java:1726)
        at org.omg.CORBA.portable.ObjectImpl._request(ObjectImpl.java:245)
        at com.ibm.WsnOptimizedNaming._NamingContextStub.get_compatibility_level
(Unknown Source)
        at com.ibm.websphere.naming.DumpNameSpace.getIdlLevel(DumpNameSpace.java:300)
        at com.ibm.websphere.naming.DumpNameSpace.getStartingContext
(DumpNameSpace.java:329)
        at com.ibm.websphere.naming.DumpNameSpace.main(DumpNameSpace.java:268)
        at java.lang.reflect.Method.invoke(Native Method)
        at com.ibm.ws.bootstrap.WSLauncher.main(WSLauncher.java:163)
```

The reasons for the mismatch are explained in the exception. You can make the corrections when you configure the outbound configuration for this server, or when you configure the inbound configuration of the downstream server. If multiple reasons exist for a failure, the reasons are explained as message text in the exception.

Typically, the outbound authentication configuration is for an upstream server to communicate with a downstream server. Most likely, the upstream server is a servlet server and the downstream server is an Enterprise JavaBeans (EJB) server. On a servlet server, the client authentication that is performed to access the servlet can be one of many different types of authentication, including client certificate and basic authentication. When receiving basic authentication data, whether through a prompt login or a form-based login, the basic authentication information is typically authenticated to from a credential of the mechanism type that is supported by the server, such as Lightweight Third Party Authentication (LTPA) or LocalOS. 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, click **Security** > **Global Security**. Under Authentication, click **Authentication protocols > CSIv2 outbound 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 Outbound Authentication**.

Authentication features include the following layers of authentication that you can use simultaneously:

Transport layer
The transport layer, the lowest layer, might contain a Secure Sockets Layer (SSL) client certificate as the identity.

Message layer
The message layer might contain a user ID and password or authenticated token.

Attribute layer
The attribute layer might contain an identity token, which is an identity from an upstream server that is already authenticated. The attribute layer has the highest priority, followed by the message layer and then the transport layer. If this server sends all three, only the attribute layer is used by the downstream server. The only way to use the SSL client certificate as the identity is if it is the only information presented during the outbound request.

Basic authentication:

Specifies whether to send a user ID and a password from the client to the server for authentication.

This type of authentication occurs over the message layer. Basic authentication also involves delegating a credential token from an already authenticated credential, provided the credential type is forwardable (for example, Lightweight Third Party Authentication (LTPA)). Basic authentication refers to any authentication over the message layer and indicates user ID and password as well as token-based authentication.

If you select **Basic Authentication**, the following options are available:

Never This option indicates that this server does not send user ID and password authentication information to downstream servers. By selecting never, requests to downstream servers that require basic authentication fail.

Supported
This option indicates that this server can specify a user ID and password to authenticate with downstream servers. However, a method might be invoked without this type of authentication. For example, the server can use anonymous or client certificate instead.

Required
This option indicates that this server must specify a user ID and password to authenticate with downstream servers for any method request. This server cannot initiate requests with servers that do not support or require basic authentication for inbound requests.

Client certificate authentication:

Specifies whether a client certificate from the configured keystore file is used to authenticate to the server when the SSL connection is made between this server and a downstream server (provided that the downstream server supports client certificate authentication).

Typically, client certificate authentication has a higher performance than message layer authentication, but requires some additional setup. These additional steps include verifying that this server has a personal certificate and that the downstream server has the signer certificate of this server.

If you select client certificate authentication, the following options are available:

Never This option indicates that this server does not attempt Secure Sockets Layer (SSL) client certificate authentication with downstream servers.

Supported
This option indicates that this server can use SSL client certificates to authenticate to downstream

servers. However, a method can be invoked without this type of authentication. For example, the server can use anonymous or basic authentication instead.

Required
This option indicates that this server must use SSL client certificates to authenticate to downstream servers.

Identity assertion:

Specifies whether to assert identities from one server to another during a downstream enterprise bean invocation.

The identity asserted is the invocation credential that is determined by the RunAs mode for the enterprise bean. If the RunAs mode is Client, the identity is the client identity. If the RunAs mode is System, the identity is the server identity. If the RunAs mode is Specified, the identity is the identity specified. The receiving server receives the identity in an identity token and also receives the sending server identity in a client authentication token. The receiving server validates the identity of the sending server to ensure a trusted identity.

When specifying identity assertion on the CSIv2 Authentication Outbound panel, you must also select basic authentication as supported or required on the CSIv2 Authentication Outbound panel. The server identity can then be submitted with the identity token, so that the receiving server can *trust* the sending server. Without specifying basic authentication as supported or required, trust is not established and the identity assertion fails.

Stateful sessions:

Specifies whether to reuse security information during authentication. This option is usually used to increase performance.

The first contact between a client and server must fully authenticate. However, all subsequent contacts with valid sessions reuse the security information. The client passes a context ID to the server, and that ID is used to look up the session. The context ID is scoped to the connection, which guarantees uniqueness. When the security session is not valid and if authentication retry is enabled, which is the default, the client-side security interceptor invalidates the client-side session and resubmits the request transparently. For example, if the session does not exist on the server; the server fails and resumes operation.

When this value is disabled, every method invocation must authenticate again.

Login configuration:

Specifies the type of system login configuration that is used for outbound authentication.

You can add custom login modules before or after this login module by 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 **Global security** > **CSIv2 Outbound Authentication** > **Additional Settings**. You can also view this administrative console page by clicking **Servers** > **Application Servers** > *server_name* > **Server Security** > **CSIv2 Outbound Authentication** > **Additional Settings**.

Client authentication type:

Specifies the type of client authentication that is supported for outbound requests.

Data type	String
Default	SAFUSERIDPASSWORD

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 Secure Authentication Service (SAS) have the ability to configure the transport.

However, the following differences between the two protocols exist:
- CSIv2 is much more flexible than SAS, which requires Secure Sockets Layer (SSL); CSIv2 does not require SSL.
- SAS does not support SSL client certificate authentication, while CSIv2 does.
- CSIv2 can require SSL connections, while SAS only supports SSL connections.
- SAS always has two listener ports open: TCP/IP and SSL.
- CSIv2 can have as few as one listener port and as many as three listener ports. You can open one port for just TCP/IP or when SSL is required. You can open two ports when SSL is supported, and open three ports when SSL and SSL client certificate authentication is supported.

Complete the following steps to configure the Inbound transport panels in the administrative console:

1. Click **Security > 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:
   - CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS - CSIv2 Client Authentication SSL Port
   - CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS - CSIv2 SSL Port
   - SAS_SSL_SERVERAUTH_LISTENER_ADDRESS - SAS SSL Port
   - ORB_LISTENER_PORT - TCP/IP Port

   For an application server, click **Servers > Application servers >***server_name* . Under Communications, click **Ports**. The Ports panel is displayed for the specified server.

   The Object Request Broker (ORB) on WebSphere Application Server uses a listener port for Remote Method Invocation over the Internet Inter-ORB Protocol (RMI/IIOP) communications, 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 > SAS inbound transport** to select the SSL settings used for inbound requests from SAS clients. Remember that the SAS protocol is used to interoperate with previous releases. When configuring the keystore and truststore files in the SSL configuration, these files need the right information for interoperating with previous releases of WebSphere Application Server. For example, a previous release has a different truststore file than the Version 5 release. If you use the Version 5 keystore file, add the signer to the truststore file of the previous release for those clients connecting to this server.

The inbound transport configuration is complete. With this configuration, you can configure a different transport for inbound security versus outbound security. For example, if the application server is the first server that is used by users, the security configuration might be more secure. When requests go to back-end enterprise bean servers, you might lessen the security for performance reasons when you go outbound. With this flexibility you can design the right transport infrastructure to meet your needs.

When you finish configuring security, perform the following steps to save, synchronize, and restart the servers:
1. Click **Save** in the administrative console to save any modifications to the configuration.

2. Stop and restart all servers, when synchronized.

Common Secure Interoperability Version 2 transport inbound settings:

Use this page to specify which listener ports to open and which Secure Sockets Layer (SSL) settings to use. These specifications determine which transport a client or upstream server uses to communicate with this server for incoming requests.

To view this administrative console page, click **Security > Global security**. 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* > **End Points**.

Provide a fixed port number for the following ports. A zero port number indicates that a dynamic assignment is made at run time.

CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS
CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS
SAS_SSL_SERVERAUTH_LISTENER_ADDRESS

Default:	SSL-Supported
Range:	TCP/IP, SSL Required, SSL-Supported

Secure Authentication Service inbound transport settings:

Use this page to specify transport settings for connections that are accepted by this server using the Secure Authentication Service (SAS) authentication protocol. The SAS protocol is used to communicate securely to enterprise beans with previous releases of the WebSphere Application Server.

To view this administrative console page, click **Security > Global security**. Under Authentication, click **Authentication protocol > SAS inbound transport**.

SSL Settings:

Specifies a list of predefined SSL settings to choose from for inbound connections. These settings are configured at the SSL Repertoire panel.

Data type:	String
Default:	DefaultSSLSettings

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 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** .

   This panel includes the SSL configuration of keystore files, truststore files, file formats, security levels, ciphers, cryptographic token selections, and so on. 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.
3. Select the SSL settings used for outbound requests to downstream Secure Authentication Service (SAS) servers. Click **Security > Global security**. Under Authentication, click **Authentication Protocol > SAS Outbound transport**. Remember that the SAS protocol allows interoperability with previous releases. When configuring the keystore and truststore files in the SSL configuration, these files have the correct information for interoperating with previous releases of WebSphere Application Server. For example, a previous release has a different personal certificate than the Version 5 release. If you use

the keystore file from the Version 5.0 release, you must add the signer to the truststore file of the previous release. Also, you must extract the signer for the Version 5.0 release and import that signer into the truststore file of the previous release.

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, click **Security > Global security**. Under Authentication, click **Authentication > Authentication protocol > CSIv2 outbound transport**.

Transport:

Specifies whether the client processes connect to the server using one of the server-connected transports.

You can choose to use either SSL, TCP/IP, or Both as the outbound transport that a server supports. If you specify `TCP/IP`, the server supports only TCP/IP and cannot initiate SSL connections with downstream servers. If you specify `SSL-supported`, this server can initiate either TCP/IP or SSL connections. If you specify `SSL-required`, this server must use SSL to initiate connections to downstream servers. When you do specify SSL, decide which set of SSL configuration settings you want to use for the outbound configuration.

This decision determines which keyfile and trustfile to use for outbound connections to downstream servers.

Consider the following options:

TCP/IP
If you select this option, the server opens TCP/IP connections with downstream servers only.

SSL-required
If you select this option, the server opens SSL connections with downstream servers.

SSL-supported
If you select this option, the server opens SSL connections with any downstream server that supports them and opens TCP/IP connections with any downstream servers that do not support SSL.

Default:	SSL-supported
Range:	TCP/IP, SSL-required, SSL-supported

SSL settings:

Specifies a list of predefined SSL settings for outbound connections. These settings are configured at the SSL Configuration Repertoires panel. To access the panel, click **Security > SSL**.

Data type:	String

Default:	DefaultSSLSettings
Range:	Any SSL settings that are configured in the SSL Configuration Repertoires panel

Secure Authentication Service outbound transport settings:

Use this page to specify transport settings for connections that are accepted by this server using the Secure Authentication Service (SAS) authentication protocol. Use the SAS protocol to communicate securely to enterprise beans with previous releases of WebSphere Application Server.

To view this administrative console page, click **Security > Global security**. Under Authentication, click **Authentication protocol > SAS outbound transport**.

SSL settings:

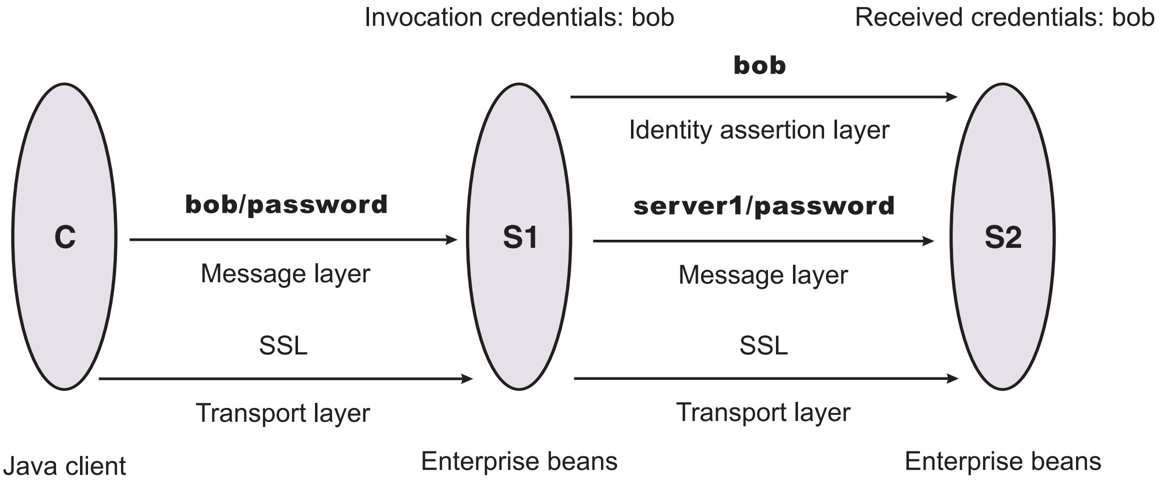
Specifies a list of predefined Secure Sockets Layer (SSL) settings to choose from for outbound connections. These settings are configured at the SSL Repertoire panel.

Data type:	String
Default:	DefaultSSLSettings

Example: Common Secure Interoperability Version 2 scenarios:

The articles included in this section are intended to demonstrate how to configure specific Common Secure Interoperability Version 2 (CSIv2) configuration examples.

Scenario 1: Basic authentication and identity assertion:



This example presents a pure Java client, C, that accesses a secure enterprise bean on server, S1, through user ″bob.″ The enterprise bean code on S1 accesses another enterprise bean on server, S2. This configuration uses identity assertion to propagate the identity of ″bob″ to the downstream server, S2. S2 trusts that ″bob″ already is authenticated by S1 because it trusts S1. To gain this trust, the identity of S1 also flows to S2 simultaneously and S2 validates the identity by checking the trustedPrincipalList list to verify that it is a valid server principal. S2 also authenticates S1. The following steps take you through the configuration of C, S1, and S2.

Configuring client, C

Client C requires message layer authentication with a Secure Sockets Layer (SSL) transport. To accomplish this task:

1. Point the client to the `sas.client.props` file using the `com.ibm.CORBA.ConfigURL=file:/C:/was/properties/sas.client.props` property.

   All further configuration involves setting properties within this file.
2. Enable SSL.

   In this case, SSL is supported but not required:
   `com.ibm.CSI.performTransportAssocSSLTLSSupported=true`,
   `com.ibm.CSI.performTransportAssocSSLTLSRequired=false`
3. Enable client authentication at the message layer.

   In this case, client authentication is supported but not required:
   `com.ibm.CSI.performClientAuthenticationRequired=false`,
   `com.ibm.CSI.performClientAuthenticationSupported=true`
4. Use all of the remaining defaults in the `sas.client.props` file.

Configuring server, S1

In the administrative console, server S1 is configured for incoming requests to support message-layer client authentication and incoming connections to support SSL without client certificate authentication. Server S1 is configured for outgoing requests to support identity assertion.

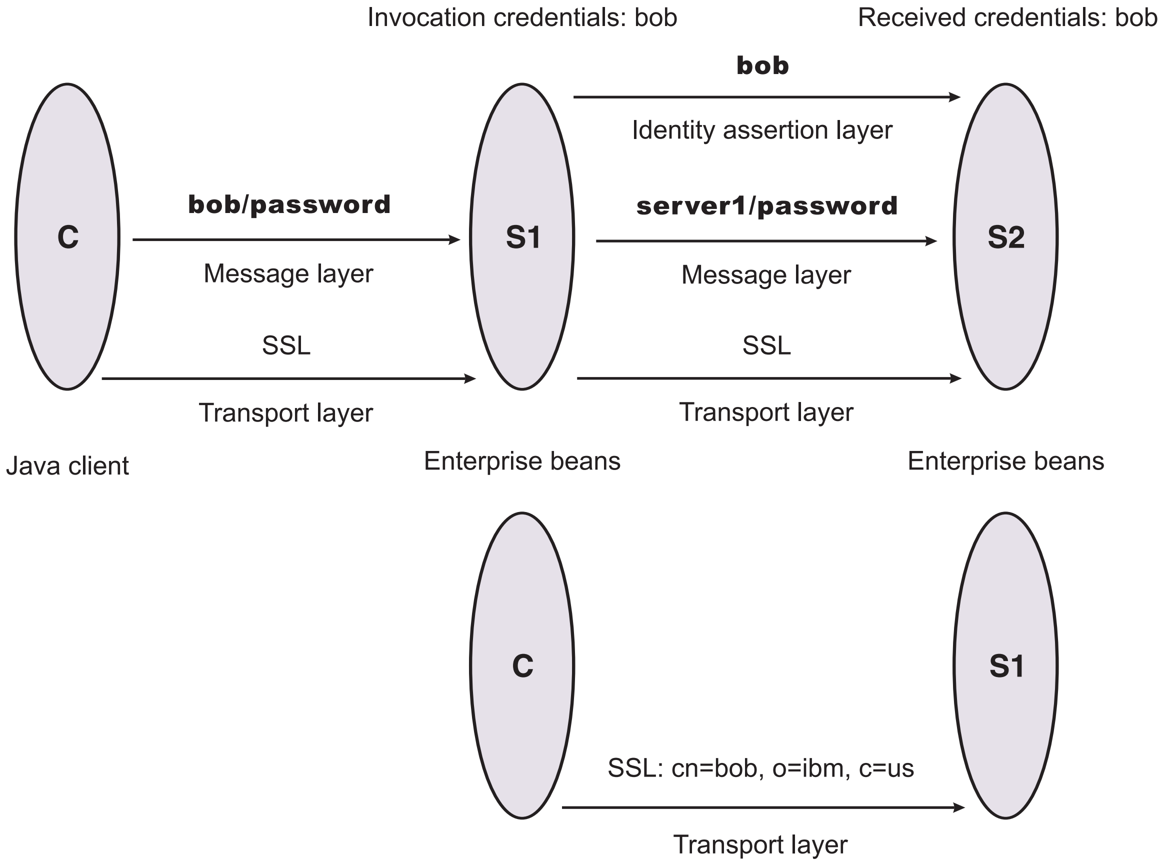
1. Configure S1 for incoming connections.
   a. Disable identity assertion.
   b. Enable user ID and password authentication.
   c. Enable SSL.
   d. Disable SSL client certificate authentication.
2. Configure S1 for outgoing connections.
   a. Enable identity assertion.
   b. Disable user ID and password authentication.
   c. Enable SSL.
   d. Disable SSL client certificate authentication.

Configuring server, S2

In the administrative console, server S2 is configured for incoming requests to support identity assertion and to accept SSL connections. Complete the following steps to configure incoming connections. Configuration for outgoing requests and connections are not relevant for this scenario.

1. Enable identity assertion.
2. Disable user ID and password authentication.
3. Enable SSL.
4. Disable SSL client authentication.

Scenario 2: Basic authentication, identity assertion, and client certificates:



This scenario is the same as Scenario 1, except for the interaction from client C2 to server S2. Therefore, the configuration of Scenario 1 still is valid, but you have to modify server S2 slightly and add a configuration for client C2. The configuration is not modified for C1 or S1.

Configuring client C2

Client C2 requires transport layer authentication (Secure Sockets Layer (SSL) client certificates). To configure transport layer authentication:

1. Point the client to the `sas.client.props` file using the `com.ibm.CORBA.ConfigURL=file:/C:/was/properties/sas.client.props` property.

   All further configuration involves setting properties within this file.
2. Enable SSL.

   In this case, SSL is supported but not required:

   ```
   com.ibm.CSI.performTransportAssocSSLTLSSupported=true,
   com.ibm.CSI.performTransportAssocSSLTLSRequired=false
   ```
3. Disable client authentication at the message layer.

   ```
   com.ibm.CSI.performClientAuthenticationRequired=false,
   com.ibm.CSI.performClientAuthenticationSupported=false
   ```
4. Enable client authentication at the transport layer where it is supported, but not required:

   ```
   com.ibm.CSI.performTLClientAuthenticationRequired=false,
   com.ibm.CSI.performTLClientAuthenticationSupported=true
   ```

Configuring server, S2

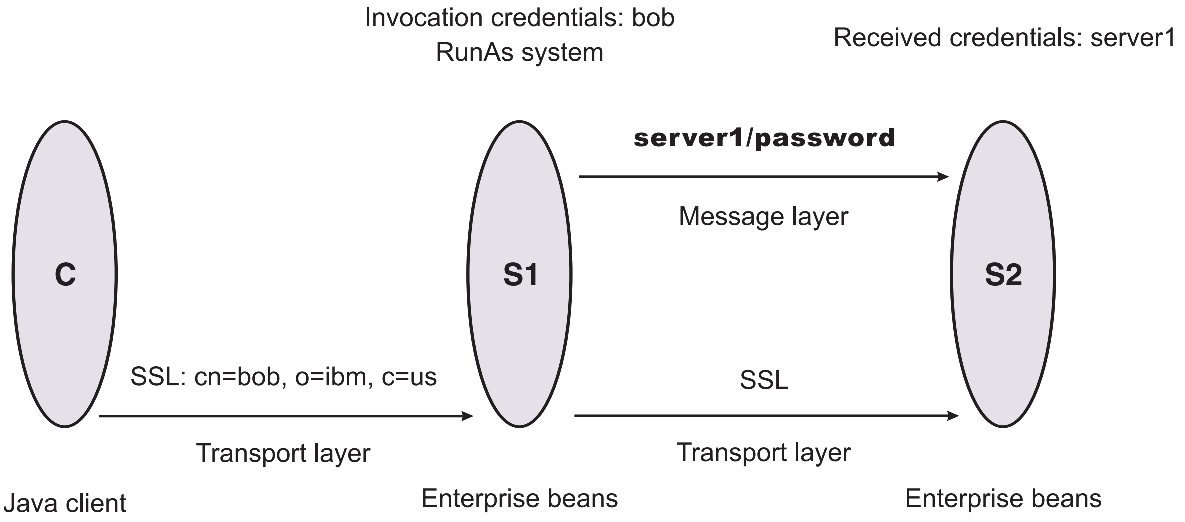
In the administrative console, server S2 is configured for incoming requests to SSL client authentication and identity assertion. Configuration for outgoing requests is not relevant for this scenario.

1. Enable identity assertion.
2. Disable user ID and password authentication.
3. Enable SSL.
4. Enable SSL client authentication.

You can mix and match these configuration options. However, a precedence exists as to which authentication features become the identity in the received credential:

1. Identity assertion
2. Message-layer client authentication (basic authentication or token)
3. Transport-layer client authentication (SSL certificates)

Scenario 3: Client certificate authentication and RunAs system:



This example presents a pure Java client, C, accessing a secure enterprise bean on S1. C authenticates to S1 using Secure Sockets Layer (SSL) client certificates. S1 maps the common name of the distinguished name (DN) in the certificate to a user in the local registry. The user in this case is `bob`. The enterprise bean code on S1 accesses another enterprise bean on S2. Because the RunAs mode is `system`, the invocation credential is set as server1 for any outbound requests.

Configuring C

C requires transport layer authentication (SSL client certificates):

1. Point the client to the `sas.client.props` file using the `com.ibm.CORBA.ConfigURL=file:/C:/was/properties/sas.client.props` property.

   All further configuration involves setting properties within this file.
2. Enable SSL.

   In this case, SSL is supported but not required: `com.ibm.CSI.performTransportAssocSSLTLSSupported=true`, `com.ibm.CSI.performTransportAssocSSLTLSRequired=false`
3. Disable client authentication at the message layer:`com.ibm.CSI.performClientAuthenticationRequired=false`, `com.ibm.CSI.performClientAuthenticationSupported=false`

4. Enable client authentication at the transport layer. It is supported, but not required:
   ```
   com.ibm.CSI.performTLClientAuthenticationRequired=false,
   com.ibm.CSI.performTLClientAuthenticationSupported=true
   ```

Configuring S1

In the administrative console, S1 is configured for incoming connections to support SSL with client certificate authentication. The S1 server is configured for outgoing requests to support message layer client authentication.

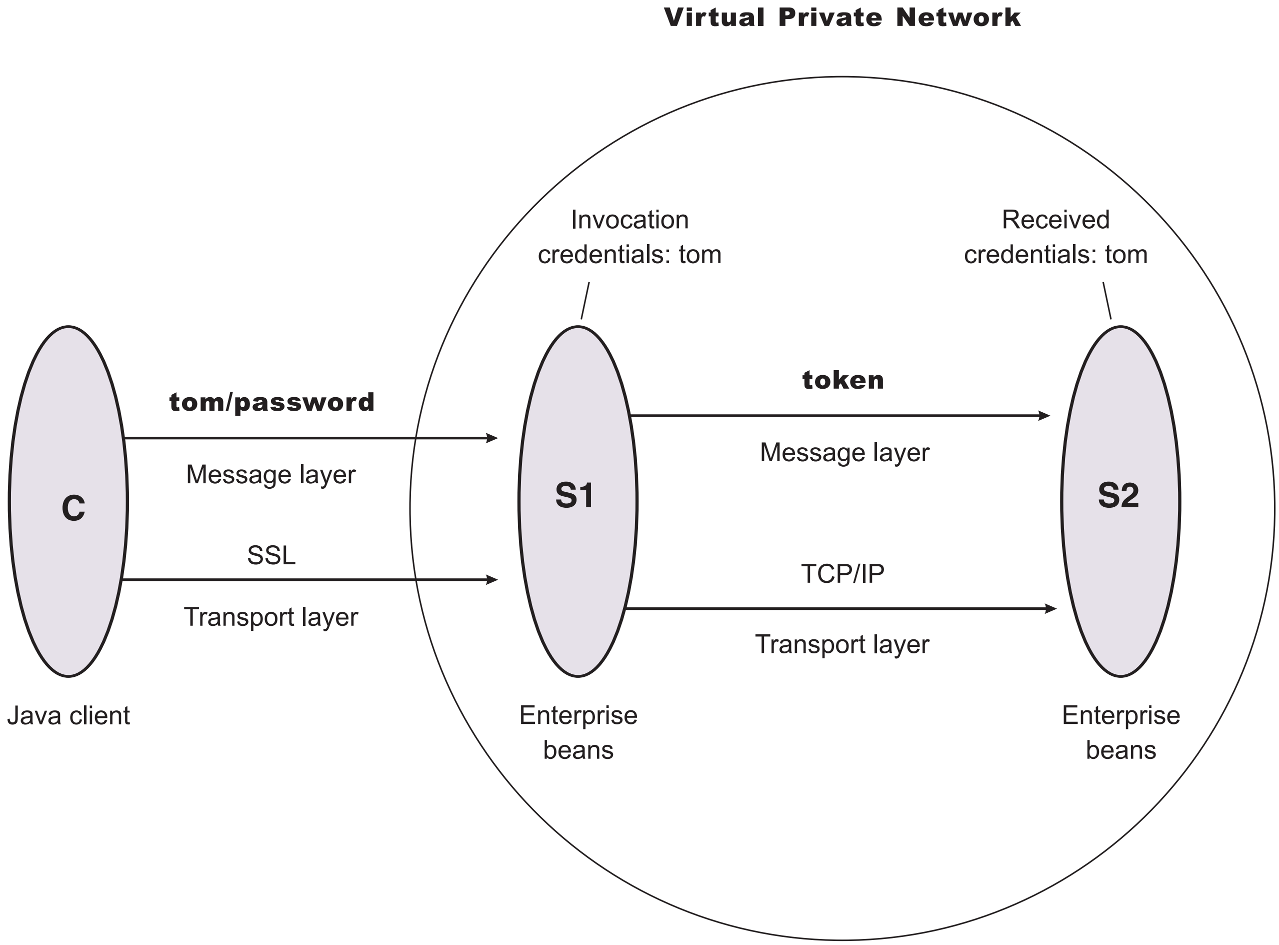
1. Configure S1 for incoming connections:
   a. Disable identity assertion.
   b. Disable user ID and password authentication.
   c. Enable SSL.
   d. Enable SSL client certificate authentication.
2. Configure S1 for outgoing connections:
   a. Disable identity assertion.
   b. Disable user ID and password authentication.
   c. Enable SSL.
   d. Enable SSL client certificate authentication.

Configuring S2

In the administrative console, the S2 server is configured for incoming requests to support message layer authentication over SSL. Configuration for outgoing requests is not relevant for this scenario.

1. Disable identity assertion.
2. Enable user ID and password authentication.
3. Enable SSL.
4. Disable SSL client authentication.

Scenario 4: TCP/IP transport using a virtual private network:



This scenario illustrates the ability to choose TCP/IP as the transport when it is appropriate. In some cases, when two servers are on the same virtual private network (VPN), it can be appropriate to select TCP/IP as the transport for performance reasons because the VPN already encrypts the message.

Configuring C

C requires message layer authentication with an SSL transport:

1. Point the client to the `sas.client.props` file using the `com.ibm.CORBA.ConfigURL=file:/C:/was/properties/sas.client.props` property. All further configuration involves setting properties within this file.
2. Enable SSL. In this case, SSL is supported but not required: `com.ibm.CSI.performTransportAssocSSLTLSSupported=true`, `com.ibm.CSI.performTransportAssocSSLTLSRequired=false`
3. Enable client authentication at the message layer. In this case, client authentication is supported but not required: `com.ibm.CSI.performClientAuthenticationRequired=false`, `com.ibm.CSI.performClientAuthenticationSupported=true`
4. Use the remaining defaults in the `sas.client.props` file.

Configuring the S1 server

In the administrative console, the S1 server is configured for incoming requests to support message-layer client authentication and incoming connections to support SSL without client certificate authentication. The S1 server is configured for outgoing requests to support identity assertion.

1. Configure S1 for incoming connections:

   a. Disable identity assertion.
   b. Enable user ID and password authentication.
   c. Enable SSL.
   d. Disable SSL client certificate authentication.
2. Configure S1 for outgoing connections:
   a. Disable identity assertion.
   b. Enable user ID and password authentication.
   c. Disable SSL.

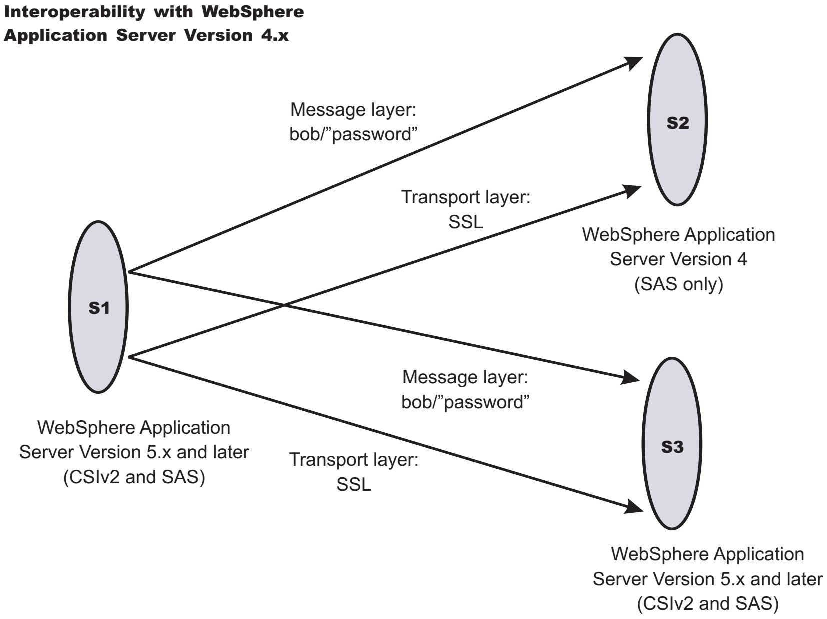
It is possible to enable SSL for inbound connections and disable SSL for outbound connections. The same is true in reverse.

Configuring the S2 server

In the administrative console, the S2 server is configured for incoming requests to support identity assertion and to accept SSL connections. Configuration for outgoing requests and connections are not relevant for this scenario.

1. Disable identity assertion.
2. Enable user ID and password authentication.
3. Disable SSL.

Scenario 5: Interoperability with WebSphere Application Server Version 4.x:


Interoperability with WebSphere Application Server Version 4.x
Message layer: bob/"password"
S2
Transport layer: SSL
WebSphere Application Server Version 4 (SAS only)
S1
Message layer: bob/"password"
WebSphere Application Server Version 5.x and later (CSIv2 and SAS)
Transport layer: SSL
S3
WebSphere Application Server Version 5.x and later (CSIv2 and SAS)

The purpose of this scenario is to show how secure interoperability can occur between different releases simultaneously while using multiple authentication protocols (Security Authentication Service (SAS) and Common Secure Interoperability Version 2 (CSIv2)). For WebSphere Application Server Version 5.x or later to communicate with a WebSphere Application Server Version 4, Version 5.x or later server must support either SAS or SAS and CSIv2 as the protocol choice. By choosing SAS and CSIv2, the Version 5.x or later server also can communicate with other Version 5.x or later servers that support CSI. If the only servers in your security domain are version 5.x or later, it is recommended that you choose CSI as the protocol because this prevents the SAS interceptors from loading. However, a chance exists that any server has to communicate with a previous release of WebSphere Application Server, select the protocol choice of SAS and CSIv2.

Configuring the S1 server

The S1 server requires message layer authentication with an SSL transport. The protocol for the S1 server must be SAS and CSIv2. Configuration for incoming requests for the S1 server is not relevant for this scenario. To configure the S1 server for outgoing connections:

1. Disable identity assertion.
2. Enable user ID and password authentication.
3. Enable Secure Sockets Layer (SSL).
4. Disable SSL client certificate authentication.
5. Set authentication protocol to SAS and CSIv2 in the global security settings.

Configuring the S2 server

All previous releases of WebSphere Application Server support the SAS authentication protocol only. No special configuration steps are needed other than enabling global security on the server (S2).

Configuring the S3 server

In the administrative console, the S3 server is configured for incoming requests to support message layer authentication and to accept SSL connections. Configuration for outgoing requests and connections are not relevant for this scenario.

1. Enable identity assertion.
2. Disable user ID and password authentication.
3. Enable SSL.
4. Disable SSL client authentication.
5. Set authentication protocol to either CSI or SAS and CSIv2.

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.

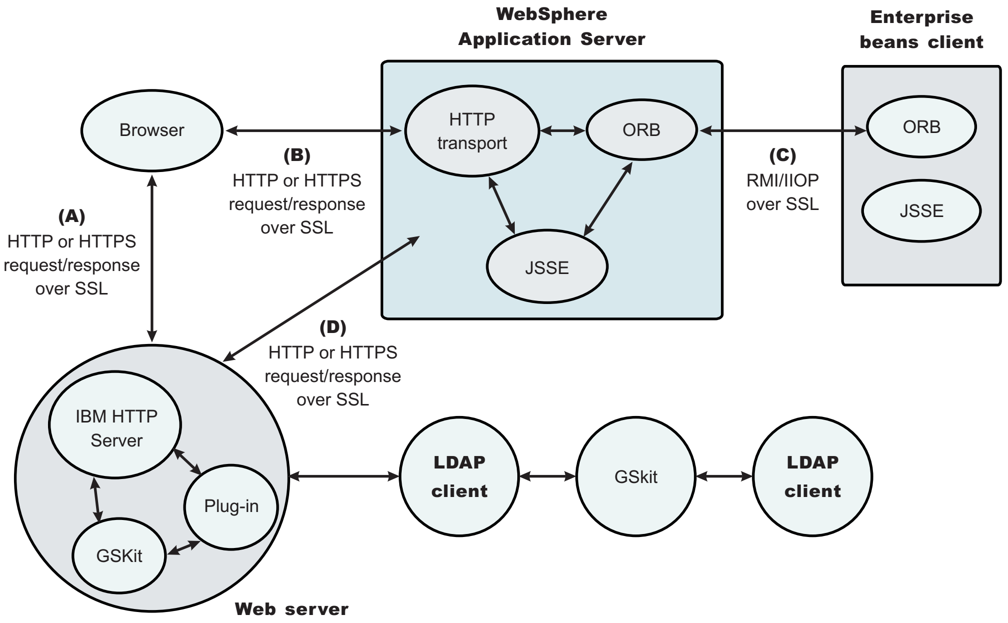


Figure 7. SSL and WebSphere Application Server

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

The SSL implementations used by WebSphere Application Server are the IBM Java Secure Sockets Extension (IBMJSSE) and the IBM Java Secure Sockets Extension 2 (IBMJSSE2). The IBMJSSE and IBMJSSE2 providers contain a reference implementation supporting SSL and Transport Layer Security (TLS) protocols and an application programming interface (API) framework. The IBMJSSE and IBMJSSE2 providers also come with a standard provider, which supplies Rivest Shamir Adleman (RSA) support for the signature-related J2EE Connector Architecture (JCA) features of the Java 2 platform, common SSL and TLS cipher suites, hardware cryptographic token device, X.509-based key and trust managers, and PKCS12 implementation for a JCA *keystore*. A graphical tool called Key Management Tool (iKeyman) also is provided to manage digital certificates. With this tool, you can create a new key database or a test digital certificate, add certificate authority (CA) roots to the database, copy certificates from one database to another as well as request and receive a digital certificate from a CA.

Note: The HTTP and JMS transports utilize the transport channel service for asynchronous I/O. This framework requires the use of IBMJSSE2 provider for SSL. Any provider you specify other than the IBMJSSE2 provider in the SSL repertoire is ignored and the IBMJSSE2 provider is used. Other SSL transports such as IIOP over SSL and LDAP over SSL utilize the provider you specify in the SSL repertoire configuration.

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.
- Only `com.ibm.crypto.provider.IBMJCE`, `com.ibm.jsse.IBMJSSEProvider`, `com.ibm.security.cert.IBMCertPath`, and `com.ibm.crypto.pkcs11.provider.IBMPKCS11` are provided as the cryptography package providers.
- Interoperability of the IBMJSSE implementation with other SSL implementations by vendors is limited to tested implementations. The tested implementations include Microsoft Internet Information Services (IIS), BEA WebLogic Server, IBM AIX, and IBM AS/400.
- Hardware token support is limited to supported cryptographic token devices. .

Tested for SSL clients	Tested for SSL clients or servers
IBM Security Kit Smartcard	IBM 4758-23
GemPlus Smartcards	IBM 4758-23
Rainbow iKey 1000/2000(USB ″Smartcard″ device)	IBM 4758-23

- 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 IBMJSSEFIPS and IBMJCEFIPS provider packages only.

Important: The IBMJSSEFIPS and IBMJCEFIPS modules are undergoing FIPS 140-2 certification. For more information on the FIPS certification process and to check the status of the IBM submission, see the 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
- SSL_RSA_WITH_NULL_MD5
- SSL_RSA_WITH_NULL_SHA
- SSL_DH_anon_WITH_AES_128_CBC_SHA *
- SSL_DH_anon_WITH_AES_256_CBC_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 *

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)

Name	Description
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 server
- Secure Lightweight Directory Access Protocol (LDAP) client.

Configuring SSL is different between client and server with WebSphere Application Server

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
   - For the Secure Authentication Services (SAS) authentication protocol only: com.ibm.CORBA.standardPerformQOPModels
   - For the cryptographic token device:
     – com.ibm.ssl.tokenType
     – com.ibm.ssl.tokenLibraryFile
     – com.ibm.ssl.tokenPassword
     – com.ibm.ssl.tokenSlot (added as a custom property)

   Note: Although WebSphere Application Server supports the IBM Federal Information Processing Standard-approved Java Secure Socket Extension (IBMJSSEFIPS), IBMJSSEFIPS is not supported for the HTTP and JMS transports due to their use of the channel framework which requires the IBMJSSE2 provider. This provider itself does not need to be FIPS compliant because it uses IBMJCE for encryption.
2. 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.
3. Create an SSL configuration repertoires alias or entry. You can select the alias later when a component is configured for SSL support. An SSL configuration repertoires entry contains the following fields:

- Typical configuration settings:
  - Alias
  - Key file name
  - Key file password
  - Key file format
  - Trust file name
  - Trust file password
  - Trust file format
  - Client authentication
  - Security level
  - Cipher suites
- For the cryptographic token device:
  - Cryptographic token (Create the alias first so you can configure these fields).
    - Token type
    - Library file
    - Password
- For additional Java properties:
  - Custom properties (Create the alias first so you can configure these fields).
    - `com.ibm.ssl.contextProvider`
    - `com.ibm.ssl.protocol`
    - `com.ibm.ssl.tokenSlot` (for crypto slot)
    - `com.ibm.ssl.keyStoreClientAlias` (alias selection for client authentication to servers)
    - `com.ibm.ssl.keyStoreServerAlias` (alias selection for server authentication to clients)

Note: WebSphere Application Server contains IBM Developer Kit for Java Technology Edition Version 1.4.x , which includes changes from IBM Developer Kit for Java Technology Edition Version 1.3. See Changes to IBM Developer Kit for Java Technology Edition Version 1.4.x for more information.

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 tool. See ″Assembling applications″ in the information center for more information. 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 tool.
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.

Refer to the Map certificates to users article to determine how a certificate is authenticated within the product.

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″ in the information center.

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 1484 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*\etc\DummyServerTrustFile.jks file using the key management utility that shipped with WebSphere Application Server. The password is WebAS.

b. Click **Personal Certificates > Import**. The **Import Key** panel is displayed. Specify LDAP_cert.arm for the file name. Complete this step for all the servers including 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 1418 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.

Configuring IBM HTTP Server for Secure Sockets Layer mutual authentication:

IBM HTTP Server supports Secure Sockets Layer (SSL) Version 2 and Version 3 and Transport Layer Security (TLS) Version 1. IBM HTTP Server is based on the Apache Web server, but for SSL configuration it requires the IBM-supplied SSL modules, rather than the OpenSSL modules. This document describes configuration of IBM HTTP Server, although it is possible to use another supported Web server.

SSL is disabled by default and it is necessary to modify a configuration file and generate a server-side certificate using the key management utility (iKeyman) provided with IBM HTTP Server to enable SSL.

1. For a single server, enable SSL on IBM HTTP Server (port 443,for example).
2. To set up certificates complete the following steps: Start the key management utility by clicking **Start > Programs > IBM HTTP Server > Start Key Management Utility**. Refer to Requesting a CA-signed personal certificate, Creating a certificate signing request (CSR), Receiving a CA-signed personal certificate, and Extracting a public certificate for use in a truststore file
3. Create a key database and click **Key Database File > New**.
4. Type a file name, `serverkey.kdb`, for example, and the location path. Click **OK**.
5. Type a password, select the **Stash the password to a file** check box and click **OK**.
6. Obtain a personal certificate for IBM HTTP Server: Click **Personal Certificate** in the key management utility menu. Click **Create > New Certificate Request**. The **Create New Key and Certificate Request** panel is displayed. Complete the following information:

   Key label

   `Server_Cert`

   Key size

   Select either a 512 or a 1024 bit size for your key.

   Common name

   droplet.austin.ibm.com

   Organization

   `IBM`

   Organization unit

   WebSphere

   Locality

   Austin

   State Texas

   Zip code

   76758

   Country

   `US`

   File name

   `Server_certreq.arm`

   The Verisign Test CA Root Certificate is in the set of signer certificates that ship with the IKeyMan utility for IBM HTTP Server.
7. Go to `http://www.verisign.com`, click **Free SSL Trial**. Complete the profile information, click **Submit**, and click **Continue** twice.
8. Use your favorite text editor to edit the request file `Server_certreq.arm`, and copy the entire contents of the file into the browser request panel. Click **Continue**. VeriSign sends the signed personal certificate to your e-mail.
9. Copy and paste this certificate into a file, for example `Server_Cert.arm`. Click **Personal Certificate** from the menu in the key management utility. Click **Receive**. Specify the file name, `Server_Cert.arm`,

and click **OK**. You might need to add VeriSign test root certificate to the signer certificates for the receive to be successful. Close the `serverkey.kdb` file.

10. To allow IBM HTTP Server to support HTTPS, port 443, for example, enable SSL on IBM HTTP Server. Modify the configuration file of IBM HTTP Server, `IHS_HOME/conf/httpd.conf`. You also can enable SSL through the IBM HTTP Server administrative console. Open the `IHS_HOME/conf/httpd.conf`file and add the following lines to the bottom of the file:

    ```
    LoadModule  ibm_ssl_module   modules/mod_ibm_ssl.so
    Listen 443
    <VirtualHost  droplet.austin.ibm.com:443>
    ServerName  droplet.austin.ibm.com
    DocumentRoot <install_root>\htdocs
    SSLEnable
    #SSLClientAuth  required
    </VirtualHost>
    SSLDisable
    Keyfile <IHS_HOME>/serverkey.kdb
    ```

    Note: Change the host name and the path for the key file accordingly. Modify the Web server to support client certificates by uncommenting the SSLClientAuth directive shown in the `httpd.conf` file.

    SSLClientAuth required
11. Restart IBM HTTP Server.
12. Test SSL between a browser and IBM HTTP Server. For more information on the default IBM HTTP Server port number, see Port number settings in WebSphere Application Server versions.
13. Follow the prompts to select a personal certificate if the SSLClientAuth directive is set to required.
14. To enable the application server to communicate with IBM HTTP Server using port 443, add the host alias on the default_host. In the administrative console, click **Environment > Virtual Hosts >***default_host*. Under Additional properties, click **Host Aliases > New**. Enter the following information in the appropriate fields:

    Host name
    *

    Port `443`
15. Click **Apply** and **Save** When you click **Save**, the information is written to the `security.xml` file and the Web server plug-in is automatically updated.
16. Restart WebSphere Application Server.
17. Test your connection.

You can connect to the Snoop servlet.

Enable Secure Sockets Layer communication between IBM HTTP Server and WebSphere Application Server.

Configuring the Web server plug-in for Secure Sockets Layer:

WebSphere has an internal HTTP transport which accepts HTTP requests. If you install an external HTTP server, the Web server plug-in must forward requests from the external HTTP server to WebSphere's internal HTTP transport. You should follow HTTP vendor's instruction to install and configure your HTTP server. Test your HTTP server by accessing http://your-host-URL and https://your-host-URL. You should also have Web server plugin installed. See ″Installing IBM HTTP Server″ for instructions on installing HTTP Server and Web server plugin. The connection between external HTTP server and WebSphere is by default not secured, even when global security is enabled.

This section documents the configuration necessary to instantiate a secure connection between the Web server plug-in and the internal HTTP transport in the WebSphere Application Server Web container on a distributed platform. By default, this connection is not secure, even when global security is enabled. This document discusses the configuration for IBM HTTP Server; however, the Web server-related configuration in this situation is not specific to any distributed platform Web server.

1. “Creating self-signed personal certificates” on page 1718. The Web server plug-in requires a key ring file to store its own private and public key files and to store the public certificate from the Web container key file. The following steps are required to generate a self-signed certificate for the Web server plug-in.

   When you install Web server plugin, a default key ring, plugin-key.kdb, is installed in plugin_root\etc. Use this file instead of creating a new one. In the following steps, a new file is created, but the steps are similar if you use an existing file. Create a directory on the Web server host for storing the key ring file that is referenced by the plug-in and associated files (for example, IHS_install_root\conf\keys).

   a. Create a directory on the Web server host for storing the key ring file that is referenced by the plug-in and associated files, for example: `IHS_install_root\conf\keys`.
   b. Launch the key management utility (iKeyman), which is available in the WebSphere Application Server *install_root*`\bin` installation directory.
   c. From the iKeyman menu, click **Key Database File** > **New**.
   d. Enter the following settings:

      Key database type
      `CMS Key Database File`

      File name
      `WASplugin.kdb`

      Location
      `C:\http1324\conf\keys\` or the file of your choice
   e. Click **OK**.
   f. Set the password of your choice at the password prompt and confirm the password.
   g. Click the **Stash the password to a file?** option.
   h. Click **OK**.
   i. From the iKeyman menu, click **Create > New Self-Signed Certificate** to create a new self-signed certificate key pair. Specify the following options. Optionally, you can choose to complete all of the remaining fields.

      Key label
      `WASplugin`

      Version
      `X509 V3`

      Key size
      `1024`

      Common name
      `droplet.austin.ibm.com`

      Organization
      `IBM`

      Country
      `US`

      Validity period
      `365`
   j. Click **OK**.
   k. Extract the public self-signed certificate key. This key is used later by the embedded HTTP server peer to authenticate connections that originate from the plug-in.
   l. Click **Personal Certificates** in the menu and select the `WASplugin` certificate that you just created.
   m. Click **Extract Certificate**. Extract the certificate to a file:

Data type
`Base64-encoded ASCII data`

Certificate file name
`WASpluginPubCert.arm`

Location
`C:\http1324\conf\keys` , or a directory of your choice

n. Click **OK**.

o. Close the key database and exit the iKeyman utility when you finish.

2. Generate a self-signed certificate for the Web container.

a. Launch the JKS-capable iKeyman version that is located the product `/bin` directory.

b. Click **Key Database File > New** from the iKeyman menu.

c. Enter the following settings:

Key database type
`JKS`

File name
`WASWebContainer.jks`

Location
`C:\WebSphere\AppServer\etc\` or the directory of your choice

d. Click **OK**.

e. Set the password of your choice at the password prompt and confirm the password.

f. Click **Create > New Self-Signed Certificate** from the iKeyman menu. The following values are used in this example:

Key Label
`WASWebContainer`

Version
`X509 V3`

Key size
`1024`

Common name
`droplet.austin.ibm.com`

Organization
`IBM`

Country
`US`

Validity Period
`365`

g. Click **OK**.

h. Extract the public self-signed certificate key. This key is used later by the Web server plug-in peer to authenticate connections that originate from the embedded HTTP server in the product.

i. Click **Personal Certificates** from the list. Select the **WASWebContainer** certificate that you just created. Click **Extract Certificate**. Extract the certificate to a file:

Data type
`Base64-encoded ASCII data`

Certificate file name
`WASWebContainerPubCert.arm`

Location
`C:\WebSphere\AppServer\etc\`

j. Click **OK**.

k. Close the database and exit the key management utility.

3. Exchange the public certificates.

a. Copy the `WASpluginPubCert.arm` file from the Web server machine to the WebSphere Application Server machine. The source directory in this case is `C:\http1324\conf\keys`, while the destination is `C:\WebSphere\Appserver\etc`.

b. Copy the `WASWebContainerPubCert.arm` file from the product machine to the Web server machine. The source directory in this case is `C:\WebSphere\Appserver\etc`, while the destination is `C:\http1324\conf\keys`.

4. Import the certificate into the Web server plug-in key file.

   a. On the Web server machine, launch the iKeyman utility, which supports the CMS key database format.

   b. From the iKeyman menu, click **Key Database File > Open** and select the previously created key database file: `WASplugin.kdb`.

   c. In the password prompt window, enter the password. Click **OK**.

   d. Click **Signer Certificates** from the list and click **Add**. This action imports the public certificate previously extracted from the embedded HTTP server (Web container) keystore file.

      Data type
      : `Base64-encoded ASCII data`

      Certificate file name
      : `WASWebContainerPubCert.arm`

      Location
      : `C:\WebSphere\Appserver\etc\`

   e. Click **OK**. You are prompted for a label name that represents the trusted signer public certificate.

   f. Enter a label for the certificate: `WASWebContainer`.

   g. Close the key database and exit IKeyman when you finish.

5. Import the certificate into the Web container keystore file.

   a. On the WebSphere Application Server machine, launch the JKS-capable iKeyman version, which is located in the product `/bin` directory.

   b. From the iKeyman menu, click **Key Database File > Open**. Select the previously created `WASWebContainer.jks` file.

   c. In the password prompt window, enter the password. Click **OK**.

   d. Click **Signer Certificates** from the list. Click **Add**. This action imports the public certificate previously extracted from the embedded HTTP server (Web container) keystore file.

      Data type
      : `Base64-encoded ASCII data`

      Certificate file name
      : `WASpluginPubCert.arm`

      Location
      : `C:\WebSphere\Appserver\etc\`

   e. Click **OK**. You are prompted for a label name that represents the trusted signer public certificate.

   f. Enter a label for the certificate: `WASplugin`.

   g. Close the key database and exit iKeyman when you finish.

6. Modify the Web server plug-in file. In a production environment, add the secure transport definition, port 9443, to the `plugin-cfg.xml` file. For example, your modified `plugin-key.kdb` file contains the following lines:

```
<Transport Hostname="hpws07" Port="9080" Protocol="http"/>
<Transport Hostname="hpws07" Port="9443" Protocol="https"/>
```

   After you verify that the proper `plugin-key.kdb` and `plugin-key.sth` files exist on the Web server, modify the `plugin-cfg.xml` file that resides on the Web server. You must specify the local path to both the `plugin-key.kdb` and `plugin-key.sth` files in the `plugin-cfg.xml` file.

Important: If you manually edit the `plugin-cfg.xml` file and an automatic regeneration of the file occurs, you must replace your manual edits.

7. Modify the Web container to support SSL. To complete the configuration between Web server plug-in and Web container, modify the WebSphere Application Server Web container to use the previously created self-signed certificates.
   a. Start the WebSphere Application Server administrative console.
   b. Click **Security > SSL**.
   c. Click **New JSSE repertoire** to create a new entry in the repertoire. Provide the following values to complete the form:

      Alias `WebContainerSSLSettings`

      Security level
      `HIGH`

      Key file name
      `C:\WebSphere\Appserver\etc\WASWebContainer.jks`

      Key file password
      <*key_file_password*>

      Key file format
      `JKS`

      Trust file name
      `C:\WebSphere\Appserver\etc\WASWebContainer.jks`

      Trust file password
      <*trust_file_password*>

      Trust file format
      `JKS`
   d. Click **OK**.
   e. If you want mutual SSL between the two parties, select the **Client authentication** option.
   f. Save the configuration in the administrative console.
   g. Click **Servers > Clusters >** <*cluster_name*> **> Cluster Members >** <*server_name*>.
   h. Under Container settings, click **Web container settings > Web container transport chains**.

      You can either modify the WCInboundDefaultSecure transport chain or click **New** and create a new transport chain.

      If you are modifying the WCInboundDefaultSecure transport chain, click **TCP Inbound Channel (TCP 4)**. Under related items, click **Ports**. Click **WC defaulthost secure** and modify the information in the Host and Port fields. Click **OK** and then **Save**.

      If you create a new transport chain, use the transport chain wizard, and specify a secure port number. You must add the same port number to the virtual hosts.
   i. Add a new virtual host entry by clicking **Environment > Virtual hosts > default_host**.
   j. Under Additional properties, click **Host aliases > New**.
   k. Enter a host name and specify the same port number that you specified for the transport chain.
   l. Click **OK**.
   m. Click **Save** at the top of the panel.
8. **Optional:** If you want to access the Web server plug-in from the Web server, click **Servers > Web servers**, and then click the **Generate Plug-in** option.
9. Test the secure connection. Test the secure connection by accessing a Web application on the WebSphere Application Server using port 9443. For example, `https://droplet.austin.ibm.com:9443/snoop`.
10. Import the correct certificate with public and private keys into the browser to test the secured connection, when client-side certification is required.
    a. Launch the iKeyman utility that supports the CMS key database file, on the Web server machine. The iKeyman utility is also bundled with IBM HTTP Server.

b. Open the key file for the plug-in, `C:\http1324\conf\keys\WASplugin.kdb`. Provide the password when prompted.

c. Click **WASplugin certificate**, located under the personal certificates. Click **Export**.

d. Save the certificate in PKCS12 format to a file, for example `C:\http1324\conf\keys\WASplugin.p12` . Provide a password to secure the PKCS12 certificate file.

e. Close the key file and exit iKeyman.

f. Copy the saved `WASplugin.p12` file to the client machine from where you access the product server.

g. Import the PKCS12 file into your browser. Then, access `https://`*`your_server_address`*`:9443/snoop`.

h. The browser asks which personal certificate to use for the connection. Select the certificate, and continue connecting.

i. After the browser test with direct product access is successful, test the connection through the Web server using port 9443. For example, `https://`*`your_server_address`*`:9443/snoop`.

The IBM HTTP Server plug-in and the internal Web server are configured for SSL.

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?
- Have you configured Common Secure Interoperability (CSI) authentication protocol for your target application server? Refer to "Configuring global security" on page 1418 for more details.

  Note: The Security Authentication Service (SAS) authentication protocol does not support Java client authentication with SSL transport.
- Have you configured your server to support secure transport for the CSIv2 inbound authentication protocol?
- Have you configured your server to support client authentication at the transport layer for the inbound CSI authentication protocol?
- If you are using a self-signed personal certificate, have you exported the public certificate from your client application Java keystore file or cryptographic token device?
- 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 your target Java truststore file 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.

Note: Java client authentication using digital certificates is supported only by the Common Secure Interoperability Version 2 (CSIv2) authentication protocol.

1. “Configuring Common Secure Interoperability Version 2 for Secure Sockets Layer client authentication.”
2. “Adding keystore files” on page 1700.
3. “Adding truststore files” on page 1700.
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. To specify the CSI protocol, set the property `com.ibm.CSI.protocol=csiv2`.
4. To match the SSL protocol configured with your server, set the property, `com.ibm.ssl.protocol`, accordingly.
5. 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:/c:/WebSphere/AppServer/profiles/`*profile_name*`/properties/sas.client.props`

Using the WebSphere Application Server to edit the sas.client.props file:

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.

Specify the keystore and truststore files in your configuration.

Adding keystore files:

A keystore file 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), PKCS12KS, JCEK
   - **com.ibm.ssl.keyStore** for a fully qualified path to the keystore file. The keystore file contains private keys and sometimes public keys.
   - **com.ibm.ssl.keyStorePassword** for the password to access the keystore file.
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 Configuration Repertoires**.
   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 keystore file.
   e. Type in the key file name for the path of the keystore file.
   f. Type in the key file password for the password to access the keystore file.
   g. Select the key file format for the keystore type. Range: JKS (default), PKCS12KS, 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.

Note: If the Cryptographic token field is selected and you only want to use cryptographic tokens for your keystore file, leave the Key file name field and the Key file password field blank.
- 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), PKCS12KS, JCEK, JCERACFKS.
   - **com.ibm.ssl.trustStore** for a fully qualified path to the truststore file. The truststore file contains the public keys.
   - **com.ibm.ssl.trustStorePassword** for the password to access the truststore file.
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.
   f. Type the trust file password for the password to access the truststore file.
   g. Select the trust file format for the truststore type. JKS (Default), PKCS12KS, 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)

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.

Alias:

Specifies the name of the specific SSL setting.

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.

New Secure Sockets Layer repertoire:

Use this page to specify the list of defined Secure Sockets Layer (SSL) configurations.

To view this administrative console page, click **Security** > **SSL** > **New JSSE repertoire**.

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 private keys.

Data type:	String

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 Application Server `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.

Client authentication:

Specifies whether to request a certificate from the client for authentication purposes when making a connection.

Data type:	Boolean
Default:	Disabled
Range:	Enabled or Disabled

When performing client authentication with the Internet InterORB Protocol (IIOP) for Enterprise JavaBeans (EJB) requests, click **Security** > **Global Security** > **CSIv2 Inbound or Outbound Authentication** from the left navigation pane of the administrative console. (You can also click **Servers** > **Application Servers** > *server_name* > **CSIv2 Inbound or Outbound Authentication**.) Click **SSL Client Certificate Authentication** to enable it for these requests.

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 only digital signing ciphers (no encryption) • Medium specifies only 40-bit ciphers (including digital signing) • High specifies only 128-bit ciphers (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

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

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:	
Default:	
Range:	

Transport channel name:

This name must be unique across all channels in a WebSphere Application Server environment. TCP transport channels and HTTP transport channels cannot have the same name if they reside within the same system.

Data type:	
Default:	
Range:	

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

Client authentication:

Specifies whether to request a certificate from the client for authentication purposes when making a connection.

This attribute is only valid when it is used by the Web container HTTP transport.

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

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

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.

Cryptographic token:

Specifies whether the server enables or disables cryptographic hardware and software support. The SOAP connector does not use hardware cryptography.

Data type:	Boolean
Default:	Disabled
Range:	Enabled or Disabled

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, IBMJSSE2, and the IBMJSSEFIPS predefined providers. IBMJSSEFIPS is the the IBMJSSE provider that is Federal Information Processing Standard (FIPS) certified. If you select **Custom JSSE provider** , enter a custom provider. For a custom provider, you first must enter the cipher suites through Custom properties under Additional Properties. Cipher suites and protocol values depend on the provider.

Note: You can only specify the IBMJSSE2 provider for transports using the channel framework, including HTTP and JMS. Any other provider specified causes the server to fail initialization. FIPS is not supported for these transports in WebSphere Application Server Version 6.

Protocol:

Specifies which SSL protocol to use.

If you are using a FIPS-approved JSSE such as IBMJSSEFIPS, you must select a TLS protocol. However, because the FIPS-approved JSSE providers are not backwards-compatible, a server that uses the TLS protocol cannot communicate with a client that uses an SSL protocol.

Default	SSLv3
Range	SSL, SSLv2, SSLv3, TLS, TLSv1

Key file name:

Specifies the fully qualified path to the SSL key file that contains public keys and might contain private keys.

You can create an SSL key file with the key management utility, or this file can correspond to a hardware device if one is available. In either case, this option indicates the source for personal certificates and for signer certificates unless a trust file is specified. The default SSL key files, `DummyClientKeyFile.jks` and `DummyServerKeyFile.jks`, contais a self-signed personal test certificate expiring on March 17, 2005. The test certificate is only intended for use in a test environment. The default SSL key files should never be used in a production environment because the private keys are the same on all the WebSphere Application Server installations. Refer to the Managing certificates article for information about creating and managing digital certificates for your WebSphere Application Server domain.

Data type:	String

Key file password:

Specifies the password for accessing the SSL key file.

Data type:	String

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

Trust file name:

Specifies the fully qualified path to a trust file containing the public keys.

You can create a trust file with the key management utility included in the WebSphere *bin* directory. Using the key management utility from Global Security Kit (GSKit) (another SSL implementation) does not work with the Java Secure Socket Extension (JSSE) implementation.

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 public key for the WebSphere Application Server Version 4.0 test certificates expires on January 15, 2004, and the public key for the WebSphere Application Server Version 5 test certificates and WebSphere Application Server CORBA C++ client expires on March 17, 2005. The test certificate is only intended for use in a test environment.

The public key for the WebSphere Application Server Version 6 test certificates expires on October 13, 2021.

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

Trust file password:

Specifies the password for accessing the SSL trust file.

Data type:	String

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

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.

Creating a Secure Sockets Layer repertoire configuration entry:

The first step in configuring Secure Sockets Layer (SSL) is 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 page in the administrative console, click **Security > SSL** to see the list of SSL repertoire settings.

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. If an SSL configuration alias is referenced elsewhere, but the alias is deleted from the SSL Configuration Repertoires panel, the SSL connection fails if the deleted alias is accessed.

With the SSL configuration repertoire, administrators can define SSL settings to use for making 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, which supports SSL connections. This selection simplifies the SSL configuration process because you can reuse many of these SSL configurations by specifying the alias in multiple places.

1. From the SSL Configuration Repertoire window, click **New**.
2. Enter the information needed to access the key file.
   a. Type the name of the key file, which must include the fully qualified path to the key file, in the Key File Name field. Type `safkeyring:///` if you are using a RACF key ring for the key file.
   b. Type 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.
   c. Select the format of the key file from the Key File Format menu.
3. Enter the information needed to access the trust file.
   a. Type the name of the trust file, which must include the fully qualified path to the trust file, in the Trust File Name field. Type `safkeyring:///` if you are using a RACF key ring as the trust store.
   b. Type the password needed to access the trust file in the Trust File Password field. Type `password` if you are using a RACF key ring as the trust store.
   c. Select the format of the trust file from the Trust File Format menu.
4. Select the **Client Authentication** option if this configuration supports client authentication. This selection only affects HTTP and LDAP requests.
5. Select the appropriate security level from the Security Level menu. Valid values are low, medium, and high. Low specifies digital signing ciphers only (no encryption), medium specifies 40-bit ciphers only (including digital signing), high specifies 128-bit ciphers only (including digital signing).

   If you are using a Federal Information Processing Standards (FIPS)-supported Java Secure Socket Extension (JSSE), you must select **High** from the Security Level menu.
6. Select a cipher suite from the Cipher Suites menu. Manually add the cipher suite if the preset security level does not define the required cipher. Select the **Cryptographic Token** check box if the RACF key ring contains keys or certificates that were created using the `RACDCERT` command with the ICSF keyword specified.
7. Select the **Cryptographic Token** check box if hardware or software cryptographic support is available.

   See “Configuring to use cryptographic tokens” on page 1726 for details regarding cryptographic support.

8. Indicate which JSSE provider you are using by either selecting **IBMJSSE**, **IBMJSSE2** (recommended) or **IBMJSSEFIPS** from the menu, or by typing the name of the provider. WebSphere Application Server includes the IBMJSSE, IBMJSSE2 and IBMJSSEFIPS JSSE providers. Use IBMJSSEFIPS only if you are using the Transport Layer Security (TLS) protocol and not the Secure Sockets Layer (SSL) protocol. See "Configuring Federal Information Processing Standard Java Secure Socket Extension files" for more information.

   If you are not using the predefined providers, configure the custom provider by clicking **Apply**, then **Custom Properties > New** in the Additional Properties section. After the custom provider is configured, return to the SSL Configuration Repertoires window and continue with these instructions.
9. Select an SSL or TLS protocol version.

   If you are using a FIPS-approved JSSE, you must select a TLS protocol version.
10. Click **Apply** to apply the changes.
11. If no errors occur, save the changes to the master configuration and restart the WebSphere Application Server.

    For more information on the FIPS certification process and to check the status of the IBM submission, see the Cryptographic Module Validation Program FIPS 140-1 and FIPS 140-2 Pre-validation List Web site.

You included additional SSL configuration repertoires with the default DefaultSSLSettings repertoire.

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 bean containers, and Lightweight Directory Access Protocol (LDAP) servers.

For the changes to take effect, restart the server after saving the configuration.

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, modify the `install_dir/profiles/profile_name/properties/sas.client.props` file and set the property:

   ```
   #com.ibm.security.useFIPS=false
   #com.ibm.security.useFIPS=true
   ```

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 crytpographic services for FIPS.

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.

Managing digital certificates:

Secure Sockets Layer (SSL) connections rely on the existence of *digital certificates*. A digital certificate reveals information about its owner, including their identity. During the initialization of an SSL connection, the server must present its certificate to the client for the client to determine the server identity. The client can also present the server with its own certificate for the server to determine the client identity. SSL is therefore, a means of propagating identity between components. Refer to “Configuring Secure Sockets Layer” on page 1688 and “Creating a Secure Sockets Layer repertoire configuration entry” on page 1708.

A client can trust the contents of a certificate if that certificate is digitally signed by a trusted third party. A Certificate Authority (CA) acts as a trusted third party and signs certificates on the basis of its knowledge of the certificate requestor. Complete the following steps to manage digital certificates using either the key management utility (iKeyman) or the keytool utility:

- Use the supplied key management utility. Refer to “Starting the key management utility (iKeyman)” on page 1717. There are two options for creating a new certificate.
  - Request that a CA generates the certificates on your behalf. The CA creates a new certificate, digitally signs it, and delivers it to the requester. Popular Web browsers are preconfigured to trust certificates that are signed by certain CAs. No further client configuration is necessary for a client to connect to the server through an SSL connection. Therefore, CA signed certificates are useful where configuration for each and every client that accesses the server is impractical. Refer to “Requesting certificate authority-signed personal certificates” on page 1719, “Creating certificate signing requests” on page 1719, “Receiving certificate authority-signed personal certificates” on page 1720, and “Extracting public certificates for truststore files” on page 1720.
  - Generate a self-signed certificate. This option might be the quickest and require the fewest details to create the certificate. However, the certificate is not signed by a CA. Any client that connects to this server over an SSL connection needs configuration to trust the signer of this certificate. Therefore, self-signed certificates are only useful when you can configure each of the clients to trust the certificate. It is possible in some cases to present a self-signed certificate to an untrusting client. In some Web browsers, when the certificate is received and does not match any of those listed in the client trust file, a prompt appears asking if the certificate should be trusted for the connection and added to the trust file. Refer to “Creating a keystore file” on page 1717, “Creating truststore files” on page 1721, “Adding keystore files” on page 1700, “Adding truststore files” on page 1700, “Creating self-signed personal certificates” on page 1718, and ″Importing signer certificates.″

  You must configure server-side options. The WebSphere Application Server stores the keystore information in the repository and the keystore files are referred to in the `security.xml` file. Therefore, complete all server-side configuration through the administration console. For Java clients, refer to “Configuring Secure Sockets Layer for Java client authentication” on page 1698.

- Use the command line Java utility called *keytool*. With keytool, you can create a private and public self-signed certificate key pair. For this example, the first user is *cn=rocaj*.
  1. Specify **RSA** for the private key to ensure that the *MD5 with RSA* signature algorithm is used. Not all Web browsers support the *DSA* cryptograph algorithm, which is the default when RSA is not specified. Set a password of at least six characters to protect the private key. Finally, specify the keystore file and keystore password (the option is storepass):

     ```
     ${WAS_HOME}/java/jre/bin/keytool -genkey -keyalg RSA -dname "cn=rocaj, ou=users,
     u=uk, DC=internetchaos, DC=com" -alias rocaj -keypass websphere -keystore
     testkeyring.jks -storepass websphere
     ```

     The previous three lines of code belong on one line, but were split onto three lines due to the width of the page.
  2. Create the second private and public self-signed certificate key pair in the same manner for the user *cn=amorv*.

     ```
     ${WAS_HOME}/java/jre/bin/keytool -genkey -keyalg RSA -dname "cn=amorv, ou=users,
     ou=uk, DC=internetchaos, DC=com" -alias amorv -keypass websphere -keystore
     testkeyring.jks -storepass websphere
     ```

     The previous three lines of code belong on one line, but were split onto three lines due to the width of the page.

     Now the keystore *testkeyring.jks* contains two self-signed certificates with the owner being the same as the issuer for each certificate.
  3. Verify the integrity and authenticity of the certificates by getting each certificate signed by the certificate authority.
     a. Generate the Certificate Signing Request, CSR-1 (for the first user cn=rocaj).

     ```
     ${WAS_HOME}/java/jre/bin/keytool -v certreq -alias rocaj -file rocajReq.csr
     -keypass
          websphere -keystore testkeyring.jks -storepass websphere
     ```

     The previous two lines of code belong on one line, but were split onto two lines due to the width of the page.

     b. On UNIX-based platforms, remove the end of line characters (^M) from the certificate signing request. To remove the end of line characters, type the following command:

     ```
     cat rocajReq.csr |tr -d "\r"
     ```

     c. Generate the CSR-2 (for the second user cn=amorv).

     ```
     ${WAS_HOME}/java/jre/bin/keytool -v -certreq -alias amorv  -file amorvReq.csr
     -keypass websphere -keystore testkeyring.jks -storepass websphere
     ```

     The previous two lines of code belong on one line, but were split onto two lines due to the width of the page.

     d. On UNIX-based platforms, remove the end of line characters (^M) from the certificate signing request. To remove the end of line characters, type the following command:

     ```
     cat amoryReq.csr |tr -d "\r"
     ```
  4. Use the free Test SSL certificate program offered by Thawte Consulting to sign the Certificate Signing Requests (CSRs) for this example. In each case, select the **Custom Cert** option and set the certificate format to use the default for your kind of certificate. The example also selects the **Generate an X.509v3 Certificate** option and saves the two resulting files as *rocajRes.arm* and *amorvRes.arm*, respectively.
  5. Import the CA trusted root certificate into the keystore. Copy and paste the Thawte test root certificate in BASE64-encoded ASCII data format to a file called *ThawteTestCA.arm*. Add the test root CA certificate into the keystore file with the following command:

```
${WAS_HOME}/java/jre/bin/keytool -import -alias "Thawte Test CA Root"
     -file ThawteTestCA.arm -keystore testkeyring.jks -storepass websphere
```

The previous two lines of code belong on one line, but were split onto two lines due to the width of the page.

6. Import the two certificate responses from the CA into the keystore file using the same alias name that was first given to the self-signed certificates. In this example, these alias names are *rocaj* and *amorv* respectively. Using an alternative alias name generates a new signer certificate and not a personal certificate chain.
   – Import the certificate response -1 (for the first user cn=rocaj).

   ```
   ${WAS_HOME}/java/jre/bin/keytool -import -trustcacerts -alias rocaj
        -file rocajRec.arm -keystore testkeyring.jks  -storepass websphere.
       Certificate reply was installed in keystore
   ```

   The previous three lines of code belong on one line, but were split onto three lines due to the width of the page.
   – Import the certificate response -2 (for the second user cn=amorv).

   ```
   ${WAS_HOME}/java/jre/bin/keytool -import -trustcacerts -alias amorv
        -file amorvRec.arm -keystore testkeyring.jks -storepass websphere.
        Certificate  reply was installed in keystore
   ```

   The previous three lines of code belong on one line, but were split onto three lines due to the width of the page.
7. Launch the JSSE ikeyman utility, which supports the PKCS12 format and the private key exporting associated with any certificate (the public key is also exported).
8. Open the *testkeyring.jks* keystore file and select the first certificate from the **Personal Certificates** menu.
9. Click **Export** and name the file, *rocajprivate.p12*. Export the second personal certificate and name it *amorvprivate.p12*.
10. Verify that the same root certificate of the authenticating CA is installed as a trusted authority in the browser.
11. To install either of the personal certificates into Netscape Communicator, click **Communicator > Tools > Security Info > Certificates > Yours**. Use the **Import a Certificate** option.
12. Enter a password or PIN for the communicator certificate database, when you attempt to import the certificate. Enter the password used when first initializing your certificate database. Enter the password protecting the PKCS#12 certificate file, as set when you exported the personal private and public certificate key pair in iKeyman.
13. Click **Verify** to check integrity and validity of the certificate. If you did not install the root CA certificate, your certificate fails the verification.
14. Verify that you modified your Web server to support client side certificate requests.
15. Go to the following URL: `https://`*`server_name`*`/snoop`; the Web browser prompts you to select a personal certificate when accessing a resource protected by the *SSLClientAuth* directive.
16. Select the HTTPS information displayed by the snoop servlet; you see the certificate SubjectDN matching the following: **Subject: CN=amorv, OU=users, OU=uk, DC=internetchaos, DC=com**.

- Refer to “Creating a Secure Sockets Layer repertoire configuration entry” on page 1708 to create a new SSL definition entry for WebSphere Application Server using the administrative console. Once a keystore file is configured, either by creating a self-signed certificate or by creating a certificate request and importing the reply, you can configure WebSphere Application Server to use the certificates. The product uses the certificates to establish a secure connection with a client through SSL.
- Set up the appropriate components to use the newly-defined SSL configuration. To ensure a secure connection, configure some non-WebSphere components, such as a Web server. A digital certificate is created for each component. The WebSphere Application Server owns a certificate and the Web server owns another certificate.

Refer to “Configuring IBM HTTP Server for Secure Sockets Layer mutual authentication” on page 1692.

Setting up SSL communication between the Web browser and WebSphere Application Server. Using digital signatures, you can communicate securely from the Web browser through the Web server to WebSphere Application Server. Once 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. Synchronize the configuration with all node agents (Network Deployment only).
3. Once synchronized, stop all servers and restart them.

Starting the key management utility (iKeyman):

It is recommended to read the documentation located in the http://www.ibm.com/developerworks/java/jdk/security/iKeymanDocs.zip file for further information.

WebSphere Application Server provides a graphical tool, the key management utility (iKeyman), for managing keys and certificates. With the key management utility, you can:

- Create a new key database
- Create a self-signed digital certificate
- Add certificate authority (CA) roots to the key database as a signer certificate
- Request and receive a digital certificate from a CA

To start the key management utility, complete the following steps:

1. Move to the `install_root/bin directory`.
2. Issue one of the following commands:
   - On Windows systems, `ikeyman.bat`
   - On UNIX systems, `ikeyman.sh`

A graphical user interface of the key management utility appears.

Creating a keystore file:

The keystore file is a key database file that contains both public keys and private keys. Public keys are stored as signer certificates while private keys are stored in the personal certificates. The keys are used for a variety of purposes, including authentication and data integrity. You can use both the key management utility (iKeyman) and the keytool utility to create keystore files.

Read the documentation located at http://www.ibm.com/developerworks/java/jdk/security/iKeymanDocs.zip for further information.

1. Start the iKeyman utility, if it is not already running.
2. Open a new key database file by clicking **Key Database File > New** from the menu bar.
3. Select the Key Database Type: JKS (default), PKCS12, and JCEKS. This is the *key file format* (or the value of `com.ibm.ssl.keyStoreType` property in the `sas.client.props` file) when you configure the SSL setting for your application.
4. Type in the file name and location. The full path of this key database file is used as the *key file name* (or the value of the `com.ibm.ssl.keyStore` property in the `sas.client.props` file) when you configure the SSL setting for your application.
5. Click **OK** to continue.
6. Then, type in password to restrict access to the file. This password is used as the *key file password* (or the value of `com.ibm.ssl.keyStorePassword` property in the `sas.client.props` file) when you configure the SSL setting for your application. Do not set an expiration date on the password or save the password to a file; you must then reset the password when it expires or protect the password file. This password is used only to release the information stored by the key management utility during run time.

7. Click **OK** to continue. The tool displays all of the available default signer certificates. These certificates are the public keys of the most common certificate authorities (CAs). You can add, view or delete signer certificates from this panel.

A new SSL keystore file is created.

Prepare keystore files for an SSL connection.

Specify the keystore file in the configuration of WebSphere Application Server. Create a truststore if one does not yet exist.

Creating self-signed personal certificates:

A self-signed personal certificate is a temporary digital certificate you issue to yourself, acting as the certificate authority (CA). Creating a self-signed certificate creates a private key and a public key within the key database file. The self-signed certificate is created in a keystore file and it is useful when you develop and test your application. You can also create a self-signed personal certificate from your cryptographic token device.

If you want to create a self-signed certificate for a keystore, you must have already created the keystore file. (Refer to “Creating a keystore file” on page 1717 for more information.) You can later extract the public key and add the key as a signer certificate to other truststore files.

Read the documentation in the http://www.ibm.com/developerworks/java/jdk/security/iKeymanDocs.zip file for further information about how to create a self-signed personal certificate within a key database file.

1. Start the key management utility, if it is not already running.
2. Click **Key Database file > Open** to select an existing file, or click **Key Database file > New** to select a new file. Select ″CMS″ for Key database type, select ″key.kdb″ for file name, and enter a directory for the file location.
3. Click **New Self-Signed** from the tool bar or click **Create > New Self-Signed Certificate**.
4. Select the **X509** version and the key size that suits your application.
5. Enter the appropriate information for your self-signed certificate:

   Key Label

   Give the certificate a key label, which is used to uniquely identify the certificate within the keystore file. If you have only one certificate in each keystore file, you can assign any value to the label. However, it is good practice to use a unique label related to the server name.

   Common Name

   Enter the common name. This name is the primary, universal identity for the certificate; it should uniquely identify the principal that it represents. In a WebSphere environment, certificates frequently represent server principals, and the common convention is to use common names of the form *host_name* and *server_name*. The common name must be valid in the configured user registry for the secured WebSphere environment.

   Organization

   Enter the name of your organization.

   Optional fields

   Enter the organization unit (a department or division), location (city), state and province (if applicable), zip code (if applicable), and select the two-letter identifier of the country in which the server belongs. For a self-signed certificate, these fields are optional. However, commercial CAs might require them.

   Validity period

   Specify the lifetime of the certificate in days, or accept the default.
6. Click **OK**.

Your key database file now contains a self-signed personal certificate.

Create a self-signed test certificate for testing purposes. If you need a test certificate signed by a certificate authority, follow the procedure in Creating a certification request.

Requesting certificate authority-signed personal certificates:

In a production environment, use a personal certificate signed by a certificate authority (CA). The principal or the owner of the CA-signed personal certificate is authenticated by a CA when the CA signs the principal certificate. Since the certificate authorities (CAs) keep their private keys secure, the signed certificate is more trustworthy than a self-signed certificate. Certificate authorities are entities that issue valid certificates for other entities. Well-known CAs include VeriSign, Entrust, and GTE CyberTrust. You can request a test certificate or a production certificate from some of the CAs like VeriSign.

The authentication process by a CA can take time. Commercial CAs often require up to a week to complete their authentication process. Even on-site CAs can take several minutes, if not hours, or even days, to complete their authentication process. Therefore, you must plan for the certificates that you need.

Considering the following points when you plan for the CA-signed certificate:

- On the certificate signing request that you send to the CA, specify the common name for the certificate. The common name is the primary, universal identity for the certificate. It should uniquely identify the principal that it represents. Verify that the common name is valid in the configured user registry for the WebSphere domain.
- Check the formatting of the address fields that your CA requires when planning the address for a certificate request.

1. Create and send a certificate signing request (CSR) to the CA.
2. Visit the CA Web site and follow the instructions to request a test or production certificate.

Once the request is accepted, the certificate authority verifies your identity and finally issues a signed certificate to you. The certificate is usually sent through e-mail.

Request a production certificate from a trusted CA for the production WebSphere Application Server environment. Once you receive the e-mail from the CA, follow the instructions to store your signed certificate as a file. Receive or store the certificate into the keystore file as a personal certificate.

Creating certificate signing requests:

To obtain a certificate from a certificate authority, submit a certificate signing request (CSR) using the key management utility (iKeyman). You can request either production or test certificates from a CA with a CSR. With the key management utility, generating a certificate signing request also generates a private key for the application for which the certificate is requested. The private key remains in the application keystore file, so it stays private. The public key is included in the certificate requested.

For information on how to create a certificate signing request from a key database file, see the http://www.ibm.com/developerworks/java/jdk/security/iKeymanDocs.zip file.

1. Start the key management utility, if it is not already running.
2. Open the key database file from which you want to generate the request.
3. Type the password and click **OK**.
4. Click **Create > New Certificate Request**. The Create New Key and Certificate Request window displays.
5. Type a **Key Label**, a **Common Name**, and **Organization**; and select a Country. For the remaining fields, accept the default value, type a value, or select new values. The common name must be valid in the configured user registry for the secured WebSphere environment.
6. Type in a name for the file, such as `certreq.arm`.
7. Click **OK** to complete.

8. **Optional:** On UNIX-based platforms, remove the end of line characters (^M) from the certificate signing request. To remove the end of line characters, type the following command:

   ```
   cat certreq.arm |tr -d "\r" > new_certreq.arm
   ```
9. Send the certreq.arm file to the certificate authority (CA) following the instructions from the CA Web site for requesting a new certificate.

The Personal Certificate Requests list shows the key label of the new digital certificate request you just created. Send the file to a CA to request a new digital certificate, or cut and paste the request into the request forms of the CA Web site.

You need to request a certificate authority-signed digital certificate for your secure WebSphere domain. Once you submit the certificate signing request, wait for the CA to accept the request. After the CA has verified your identity, it sends back the signed certificate usually through e-mail. Receive the signed certificate back to the keystore file from which you generated the CSR.

Receiving certificate authority-signed personal certificates:

Once the certificate signing request (CSR) is accepted, a certificate authority (CA) processes the request and verifies your identity. Once approved, the CA sends the signed certificate back through e-mail. Store the signed certificate in a keystore database file. This procedure describes how to receive the CA-signed certificate into a keystore file using the key management utility (iKeyman). You use this utility the same way for both test certificates and production certificates. The primary difference between the two certificate types is the amount of time it takes for the CA to authenticate the principal your certificate represents. Test certificates are authenticated automatically based on some simple edit checks and returned to you within a few hours. Production certificates may take several days or a week to authenticate and return to you. If the CSR request is made for the cryptographic token, the certificate must be received into that token. If the request is made for the secondary key database of the token, the certificate must be received into that database.

Receive the signed certificate from the CA through e-mail. Follow the instructions from the CA to store the certificate into a file.

Read the http://www.ibm.com/developerworks/java/jdk/security/iKeymanDocs.zip file for further information about how to receive a personal certificate into a key database file from the CA.

1. Start IKeyman, if it is not already running.
2. Open the key database file from which you generated the request.
3. Type the password and click **OK**.
4. Select **Personal Certificates** from the pull-down list.
5. Click **Receive**.
6. Click **Data type** and select the data type of the new digital certificate, such as Base64-encoded ASCII data. Select the data type that matches the CA-signed certificate. If the CA sends the certificate as part of an E-mail message, you may first need to cut and paste the certificate into a separate file.
7. Type the certificate file name and location for the new digital certificate, or click **Browse** to locate the CA-signed certificate.
8. Click **OK**.
9. Type a label for the new digital certificate and click **OK**.

The personal certificate list now displays the label you just gave for the new CA-signed certificate.

Once the CA-signed certificate is successfully received, you can extract or export the public key of the certificate to a file for distribution to the network.

Extracting public certificates for truststore files:

Use this procedure to extract a public certificate, which includes its public key, from a keystore file. If a target truststore file already contains the signer certificate of the certificate authority (CA) that signed the certificate, you do not need to extract and add the certificate to the target truststore file. However, in general, you need to complete this procedure for a self-signed certificate.

Extracting a certificate from one keystore file and adding it to a truststore file is not the same as exporting the certificate and then importing it. Exporting a certificate copies all the certificate information, including its private key, and is normally only used if you want to copy a personal certificate into another keystore file as a personal certificate.

If a certificate is self-signed, extract the certificate and its public key from the keystore file and add it to the target truststore file.

If a certificate is CA-signed, verify that the CA certificate used to sign the certificate is listed as a signer certificate in the target truststore file. The keystore file must already exist and contain the certificate to be extracted.

Read the http://www.ibm.com/developerworks/java/jdk/security/iKeymanDocs.zip file for further information about how to extract a public certificate from a key database file.

1. Start the key management utility (iKeyman), if it is not already running.
2. Open the keystore file from which the public certificate will be extracted.
3. Select **Personal Certificates**.
4. Click **Extract Certificate**.
5. Click **Base64-encoded ASCII data** under Data type.
6. Enter the **Certificate File Name** and **Location**.
7. Click **OK** to export the public certificate into the specified file.

A certificate file that contains the public key of the signed personal certificate is now available for the target truststore file.

Prepare truststore files for distributing the public keys to support the secure WebSphere domain using Secure Sockets Layer (SSL). Once the keystore and truststore files are ready, make them accessible by specifying them in your client and server configurations.

Creating truststore files:

A truststore file is a key database file that contains the public keys for target servers. The public key is stored as a signer certificate. If the target uses a self-signed certificate, extract the public certificate from the server keystore file. Add the extracted certificate into the truststore file as a signer certificate. For a commercial certificate authority (CA), the CA root certificate is added. The truststore file can be a more publicly accessible key database file that contains all the trusted certificates.

Read the documentation located at http://www.ibm.com/developerworks/java/jdk/security/iKeymanDocs.zip for further information.

1. Start the key management utility (iKeyman), if it is not already running.
2. Open a new key database file by clicking **Key Database File > New** from the menu bar.
3. Click the **Key Database Type**: `JKS(Default)`, `PKCS12`, and `JCEKS`. The key database type is the *trust file format* (or the value of the `com.ibm.ssl.trustStoreType` property in the `sas.client.props` file) when you configure the SSL setting for your application.
4. Type in the file name and location. The full path of this key database file is used as the *trust file name* (or the value of `com.ibm.ssl.trustStore` property in the `sas.client.props`) when you configure the SSL setting for your application.
5. Click **OK** to continue.

6. Type in a password to restrict access to the file. This password is used as the *trust file password* (or the value of the `com.ibm.ssl.trustStorePassword` property in the `sas.client.props` file) when you configure the SSL setting for your application. Do not set an expiration date on the password or save the password to a file. You must reset the password when it expires or protect the password file. This password is used only to release the information stored by the key management utility during run time.
7. Click **OK** to continue. The tool now displays all of the available default signer certificates. These are the public keys of the most common CAs. You can add, view or delete signer certificates from this screen.

A new SSL truststore file is created.

Prepare truststore files for an SSL connection. Specify the truststore file in the configuration of WebSphere Application Server. Create a keystore file if one does not exist.

Importing signer certificates:

A *signer certificate* is the trusted certificate entry that is usually in a truststore file. You can import a certificate authority (CA) root certificate from the CA, or a public certificate from the self-signed personal certificate of the target into your truststore file, as a signer certificate.

Read the documentation located in http://www.ibm.com/developerworks/java/jdk/security/iKeymanDocs.zip file for further information.

1. Start the key management utility (iKeyman), if it is not already running.
2. Open the truststore file. The Password Prompt window displays.
3. Type the password and click **OK**.
4. Select **Signer Certificates** from the menu.
5. Click **Add**.
6. Click **Data type** and select a data type, such as Base64-encoded ASCII data. This data type must match the data type of the importing certificate.
7. Type a certificate file name and location for the CA root digital certificate or click **Browse** to select the name and location.
8. Click **OK**.
9. Type a label for the importing certificate.
10. Click **OK**.

The **Signer Certificates** field now displays the label of the signer certificate you just added. Receive a CA root certificate or the public key from your secure target.

Map certificates to users:

Client-side certificates support access to secured resources from Web or Java clients. A client presents an X.509-compliant digital certificate to perform mutual authentication with a single sockets layer-enabled server. The product security run time attempts to map the certificate to a known user in the associated Lightweight Directory Access Protocol (LDAP) directory or the custom registry. If the certificate successfully maps to a user, then the holder of the certificate is regarded as the user in the registry and is authorized as this user. In the case of LocalOS registry, the DN is parsed and the name between the first equals (=) and comma (,) is used as the mapped name. If the DN does not contain the ″=″, the complete name is used. If there is no ″,″ in the DN, everything after the ″=″ is used as the name.

After the single sockets layer-enabled server gets the client certificate, the server needs to map the certificate to a user. WebSphere Application Server supports two techniques for mapping certificates to entries in LDAP registries:

- By exact distinguished name

- By matching attributes in the certificate to attributes of LDAP entries

1. Map by exact distinguished name (DN).

   This approach attempts to map the distinguished name (DN) associated with the **Subject** field in the certificate to an entry in the LDAP directory. If the mapping is successful, the user is authenticated and is authorized according to the privileges granted to the identity in the LDAP directory.

   The mapping is case insensitive. For example, the following two DNs match on a case-insensitive comparison:

   ```
   "cn=Smith, ou=NewUnit, o=NewCompany, c=us"
   "cn=smith, ou=newunit, o=NewCompany, c=US"
   ```

   If a match is found, authentication succeeds; if no match is found, authentication fails.

2. Map by filtering certificate attributes.

   This approach maps certificate attributes to attributes of entries in an LDAP directory. For example, you can specify that the common name (CN) attribute of the **Subject** field in the certificate must match the uid attribute of your LDAP entry. If the mapping is successful, the user is authenticated and is authorized according to the privileges granted to the identity in the LDAP directory.

   If you are matching the Subject CN field in the certificate to the uid attribute of the LDAP entry, a certificate with the Subject DN ″cn=Smith, ou=NewUnit, o=NewCompany, c=us″ matches an LDAP user entry with uid=Smith.

   To use this mapping technique, you must request certificate mapping and set up the certificate filter in the administrative console.

This specification extracts the CN field from the Subject attribute in the certificate (Smith) and creates a filter (user ID = Smith) from it. The LDAP directory is searched for a user entry that matches the filter. If an entry matches the filter, authentication succeeds.

Note: The search and match of the LDAP directory are based in part on how your LDAP directory is configured.

Changes to IBM Developer Kit for Java Technology Edition Version 1.4.x: WebSphere Application Server, Version 5.1 includes the IBM Developer Kit, Java Technology Edition Version 1.4.x, which contains changes to the IBM Developer Kit, Java Technology Edition Version 1.3.x. This document is intended to assist application developers and system administrators in understanding the changes.

Security packaging changes in IBM Developer Kit, Java Technology Edition Version 1.4.x

In IBM Developer Kit, Java Technology Edition Version 1.4.x, many of the security technologies have been included in the core of the IBM Developer Kit, Java Technology Edition Version 1.4.x. Because of the packaging changes, we are supporting specific `java.security` configurations for each platform. This document discusses the impact these `java.security` configuration changes have on each platform.

Security providers for the Windows, Linux, and AIX platforms

The Windows, Linux, and AIX platforms use all of the IBM security provider implementations, which is similar to how IBM Developer Kit, Java Technology Edition Version 1.3.x shipped. Because the security technologies in IBM Developer Kit, Java Technology Edition Version 1.3.x, were not part of the core, these technologies were shipped in the `java/jre/lib/ext` directory and provided more flexibility in implementing the technologies. Only those JSSE providers configured by WebSphere Application Server are supported.

The following list shows the providers and sequence of how these providers are supported on the Windows, Linux, and AIX platforms. Add any additional providers at the end of this list of providers. The IBMJSSE, IBMJSSE2 and IBMJSSEFIPS providers are the only SSL providers supported on these platforms. You must configure HTTP and JMS transports to use the IBMJSSE2 providers because they use the channel framework (asynchronous network I/O (NIO) APIs from Java SDK 1.4.2). The NIO APIs only work with the IBMJSSE2 provider and the channel framework.

```
security.provider.1=com.ibm.crypto.provider.IBMJCE
security.provider.2=com.ibm.jsse.IBMJSSEProvider
security.provider.3=com.ibm.security.jgss.IBMJGSSProvider
security.provider.4=com.ibm.security.cert.IBMCertPath
security.provider.5=com.ibm.crypto.pkcs11.provider.IBMPKCS11
```

Security providers for the Sun Solaris environment

In the Sun Solaris environment, by default, we are using the IBM JSSE framework classes. These classes enable you to plug-in the IBMJSSE, IBMJSSE2, and IBMJSSEFIPS providers. You must configure HTTP and JMS transports to use the IBMJSSE2 providers because they use the channel framework (asynchronous network I/O (NIO) APIs from Java SDK 1.4.2). The NIO APIs only work with the IBMJSSE2 provider and the channel framework.

The following list shows thee default provider lists for the Sun Solaris environment. Add any additional providers to the end of this list.

```
security.provider.1=com.ibm.security.jgss.IBMJGSSProvider
security.provider.2=sun.security.provider.Sun
security.provider.3=com.ibm.crypto.provider.IBMJCE
security.provider.4=com.ibm.jsse.IBMJSSEProvider
security.provider.5=com.ibm.security.cert.IBMCertPath
# security.provider.6=com.ibm.crypto.pkcs11.provider.IBMPKCS11
```

Note: You only need to uncomment the IBMPKCS11 provider when using IKeyMan to access a cryptographic token device. The WAS runtime now uses the IBMPKCS11Impl provider for cryptographic token access, instead of the IBMPKCS11 provider. To get more information on this provider, please see ″Security: Resources for Learning″ for information on this provider.

Security providers for the HP-UX platform

In the HP-UX environment, by default, IBM JSSE framework classes are used. These classes enable you to plug-in the IBMJSSE, IBMJSSE2 and IBMJSSEFIPS providers. You must configure HTTP and JMS transports to use the IBMJSSE2 providers because they use the channel framework (asynchronous network I/O (NIO) APIs from Java SDK 1.4.2). The NIO APIs only work with the IBMJSSE2 provider and the channel framework.

```
security.provider.1=com.ibm.security.jgss.IBMJGSSProvider
security.provider.2=sun.security.provider.Sun
security.provider.3=com.ibm.crypto.provider.IBMJCE
security.provider.4=com.ibm.jsse.IBMJSSEProvider
security.provider.5=com.ibm.security.cert.IBMCertPath
# security.provider.6=com.ibm.crypto.pkcs11.provider.IBMPKCS11
```

Note: You must uncomment the IBMPKCS11 provider when using IKeyMan to access a cryptographic token device. The WAS runtime now uses the IBMPKCS11Impl provider for cryptographic token access, instead of the IBMPKCS11 provider. To obtain more information about this provider, see the ″Security: Resources for Learning″ article.

Changes to the CertPath API package name

In IBM Developer Kit, Java Technology Edition Version 1.3.x, the package for CertPath APIs was javax.security.cert.*. However, in IBM Developer Kit, Java Technology Edition Version 1.4.x, the package has changed to java.security.cert.*. While your applications might still work using javax.security.cert.* due to the oldcertpath.jar packaged in `${`*`WAS_INSTALL_ROOT`*`}/java/jre/lib/ext/oldcertpath.jar` file, change your applications to use the new package name for CertPath from this point forward. In this release, either package name should work, but it is recommended that you use the correct package, which is java.security.cert.*.

Known problems with IBM Developer Kit, Java Technology Edition Version 1.4.x

For a list of known problems with the various platforms related to the IBM Developer Kit, Java Technology Edition Version 1.4.x changes, please review the release notes for WebSphere Application Server, Version 5.1.

There are some known issues with the the IBMJSSE2 provider:

- When configuring a cryptographic token device, you must use the IBMJSSE2 provider. There is a dependency on the new IBMPKCS11Impl provider for cryptographic token support. This provider can only be initialized once in a JVM, and is done programmatically by the WebSphere Application Server runtime when a cryptographic token device is configured. The user of the IBMPKCS11Impl provider in applications is not supported unless the cryptographic token device is not configured for use by WebSphere Application Server.
- FIPS support does not exist for the IBMJSSE2 provider because there currently is not an IBMJCEFIPS that can be used with IBMJSSE2.
- Any transport using the channel framework, including HTTP and JMS, must use the IBMJSSE2 provider.
- Any transport using the channel framework, including HTTP and JMS, must use the IBMJSSE2 provider.
- To use AES_256 ciphers for IBMJSSE2, you must download the JCE Unlimited Strength Jurisdiction Policy.
- IBMJSSE2 provider's HTTPS protocol handler is ″com.ibm.net.ssl.www2.protocol.Handler″. The package to add to the package handler property is ″com.ibm.net.ssl.www2.protocol″.

Cryptographic token support

A *cryptographic token* is a hardware or software device with a built-in keystore implementation. The cryptographic device is used to manage certificates stored on the cryptographic tokens (also known as *smartcards*).

Both cryptographic accelerators, where the cryptographic hardware device has no persistent key storage, and secure cryptographic hardware, where a cryptographic token generates and securely stores the private key used for Secure Sockets Layer (SSL) key exchange, are supported in the product.

Hardware cryptographic token support has changed providers in Version 6. In Version 5 and before, WebSphere Application Server used com.ibm.crypto.pkcs11.provider.IBMPKCS11 provider for hardware crypto support along with the old IBMJSSE provider for SSL. The IBMPKCS11 provider is still used when accessing hardware using IKeyMan. The IBMJSSE provider can still be used, if necessary, for SSL.

Note: To use cryptographic token devices in the Solaris Operating Environment, you must edit the `${WAS_INSTALL_ROOT}/java/jre/lib/security/java.security` file. Uncomment the line containing `com.ibm.crypto.pkcs11.provider.IBMPKCS11`. By default, the line is commented out because the algorithm MD4 is not present in the IBMPKCS11 provider.

The WebSphere Application Server runtime in Version 6 now uses the com.ibm.crypto.pkcs11impl.provider.IBMPKCS11Impl provider for hardware crypto support and the IBMJSSE2 provider for SSL. Both the IBMPKCS11Impl and IBMJSSE2 providers are initialized programmatically. The IBMPKCS11Impl provider is only initialized when hardware crypto is configured in one of the SSL repertoire configurations. Once IBMPKCS11Impl provider is configured, the IBMPKCS11 provider cannot be used in the system since only one provider can initialize a hardware crypto card in the same process.

Please see the following document for more information on the IBMPKCS11Impl provider: `http://www.ibm.com/developerworks/java/jdk/security/142/pkcs11implDocs.zip`

Please see the following document for more information on the IBMJSSE2 provider: `http://www.ibm.com/developerworks/java/jdk/security/142/jsse2docs.zip`

Opening a cryptographic token using the key management utility (iKeyman)

Verify that your cryptographic token device is installed and functions properly. Create a cryptographic token, following the instructions provided by the manual of the cryptographic device.

From your cryptographic token device documentation, identify the token library. For example, the IBM 4758 PCI Cryptographic Card uses CRYPTOKI.DLL as the PKCS#11-type token library (see `http://www.ibm.com/security/cryptocards/html/library.shtml` for details).

Read the documentation located in the http://www.ibm.com/developerworks/java/jdk/security/142/ikmuserguide.pdf file for further information about using the key management utility (iKeyman).

Important: To use iKeyMan for key management with a cryptographic token device, you must edit the ${WAS_INSTALL_ROOT}/java/jre/lib/security/java.security file. Uncomment the line containing com.ibm.crypto.pkcs11.provider.IBMPKCS11.

You can use the key management utility to open a cryptographic token. Once opened, you can manage your keys and certificates just like you do with keystore and truststore files:

- Create a self-signed digital certificate
- Extract or add both certificate authority (CA) roots and personal certificate signer certificates
- Request and receive a digital certificate from a CA

1. Start the key management utility, if it is not already running.
2. Click **Key DataBase File > Open**.
3. Click **Cryptographic Token** from the list of key database types.
4. Fill in the information for **File Name** and **Location**, or browse for the cryptographic device library.
5. Click **OK** to open the library.
6. Type in the slot number in the next panel. This is the number of the slot in which you previously created the cryptographic token.
7. Enter the password. This is the password configured for the cryptographic token that you created.

All of the personal and signer certificates are stored on the cryptographic token card. With the token open, you can create or request digital certificates and receive CA-signed certificates.

Use a cryptographic token device as a key database to manage keys and certificates for an SSL connection. Once the cryptographic token is open, you can add or delete keys and certificates. Configure the cryptographic token settings in WebSphere Application Server.

Configuring to use cryptographic tokens

You can configure cryptographic token support in both client and server configuration. To configure a Java client application, use the `sas.client.props` configuration file. By default, the `sas.client.props` 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`.

To understand how to make WebSphere Application Server (both the run time and the key management utility) work correctly with any cryptographic token device, become familiar with the Java Secure Socket Extension (JSSE) documentation available in the http://www.ibm.com/developerworks/java/jdk/security/142/jsse2docs.zip. and http://www.ibm.com/developerworks/java/jdk/security/142/ikmuserguide.pdf files.

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″ in the information center.

Note: You cannot use cryptographic token devices when you have Federal Information Processing Standard (FIPS) enabled.

Important: To use iKeyMan for key management with a cryptographic token device, you must edit the `${WAS_INSTALL_ROOT}/java/jre/lib/security/java.security` file. Uncomment the line containing `com.ibm.crypto.pkcs11.provider.IBMPKCS11`.

WebSphere Application Server Version 6 and later runtime uses the IBMPKCS11Impl provider instead of the IBMPKCS11 provider for hardware crypto support. See http://www.ibm.com/developerworks/java/jdk/security/142/pkcs11implDocs.zip for more information. Refer to the ″IBM Java PKCS 11 Implementation Provider.htm″ document located in this zip file.

Note: To use cryptographic token devices in the Solaris Operating Environment, you must edit the `${WAS_INSTALL_ROOT}/java/jre/lib/security/java.security` file. Uncomment the line containing `com.ibm.crypto.pkcs11.provider.IBMPKCS11`. By default, the line is commented out because the algorithm MD4 is not present in the IBMPKCS11 provider.

1. To configure a client to use a cryptographic token, edit the `sas.client.props` file and set the following properties. Leave the **KeyStore File Name**, **KeyStore File Password**, **TrustStore File Name**, **TrustStore File Password** fields in a Secure Sockets Layer (SSL) configuration blank (or comment out the properties com.ibm.ssl.trustStore, com.ibm.ssl.trustStorePassword, com.ibm.ssl.keyStore, and com.ibm.ssl.keyStorePassword, using a # in front of the property name) , if you want to use only cryptographic tokens as your keystore.

   com.ibm.ssl.tokenType
   Specifies the type of built-in keystore file that is implemented in the cryptographic token. (For example, `com.ibm.ssl.tokenType=PKCS\#11`). The valid values are: **PKCS\#7**, **PKCS\#11**, **PKCS\#12**, and **MSCAPI**.

   com.ibm.ssl.tokenLibraryFile
   Specifies the token file name for **PKCS#7** tokens, **PKCS#12** tokens, and the library name for **PKCS#11**, MSCAPI tokens. Make sure the cryptographic token device is installed and functions properly with a cryptographic token created.

   com.ibm.ssl.tokenPassword
   Specifies the password to unlock the cryptographic token.

2. Configure your server to use the cryptographic device. Leave the **KeyStore File Name**, **KeyStore File Password**, **TrustStore File Name**, **TrustStore File Password** fields in an SSL configuration blank, if you want to use only cryptographic tokens as your keystore. 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 specific protocol's SSL configuration.

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. Complete the information for **Token Type** to specify the type of built-in keystore file that is implemented in the cryptographic token. The valid values are: **PKCS#7**, **PKCS#11**, **PKCS#12**,or **MSCAPI**.

g. Complete the information for **Library File** to specify the path to the cryptographic device driver. Make sure the cryptographic token device is installed and functions properly with a new cryptographic token.

h. Complete the information for **Password** to specify the password for unlocking the cryptographic device.

i. Click **OK**. This returns you to the **SSL configuration repertoires - General Properties** panel for this alias.

j. Optionally, to configure a specific Token Slot for the cryptographic token device, click **Custom Properties** from the SSL configuration repertoires - General Properties panel. Add a new property name, **com.ibm.ssl.tokenSlot**, and a property value with the slot number, for example: 0. Optionally, to configure the selection of a specific inbound certificate alias (the alias selected for server transports) within the configured slot, add a new property name, **com.ibm.ssl.keyStoreServerAlias**, with a property value equal to the certificate alias name as it appears when viewing the slot through iKeyMan. Optionally, to configure the selection of a specific outbound certificate alias (the alias selected for client transports) within the configured slot, add a new property name, **com.ibm.ssl.keyStoreClientAlias**, with a property value equal to the certificate alias name as it appears when viewing the slot through iKeyMan, for example. Click **OK** to exit the Custom Properties panel and return to the SSL configuration repertoires - General Properties panel.

k. 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).

l. 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)**.

   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 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 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**.

m. 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 transferred over the network.

WebSphere Application Server uses the cryptographic token as a keystore file for and SSL connection.

If the server configuration has changed, restart the configured server.

Cryptographic token settings:

Use this page to configure cryptographic token settings. A cryptographic token is a hardware or software device with a built-in key store implementation. The cryptographic device is used to manage certificates stored on the cryptographic tokens. These devices are also known as smartcards.

The following types of cryptographic accelerators are supported by WebSphere Application Server:

- A cryptographic hardware device that is without a persistent key storage.
- Secure cryptographic hardware, where a cryptographic token generates and securely stores the private key used for Secure Sockets Layer (SSL) key exchange.

To view this administrative console page, click **Security > SSL >** *alias_name*. Under Additional Properties, click **Cryptographic token**.

Token type:

Specifies the type of built-in keystore file that is implemented in the cryptographic token, such as PKCS#11.

The WebSphere Application Server uses an implementation of Java Secure Socket Extension (JSSE) to support cryptographic token with Secure Sockets Layer (SSL). Different cryptographic devices are supported. For an SSL server, the following devices are supported:

- IBM 4758-23
- nCipher nForce
- Rainbow Cryptoswift

For an SSL client, the following devices are supported:

- IBM 4758-23
- nCipher nForce
- Rainbow Cryptoswift
- IBM Security Kit Smartcard
- GemPlus Smartcards
- Rainbow iKey 1000/2000 (USB ″Smartcard″ device)
- Eracom CSA8000

Follow the documentation that accompanies your device to install your cryptographic token.

Data type:	String

Library file:

Specifies the dynamic link library (DLL) or shared object that implements the interface to the cryptographic token device.

Data type:	String

Password:

Specifies the password for the cryptographic token device.

Data type:	String

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 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 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). You can specify the IBMJSSEFIPS provider for all transports with the exception of JMS and HTTP.

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. Once 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

You can find more information documented for the JSSE APIs if you download and unzip the http://www.ibm.com/developerworks/java/jdk/security/142/jsse2Docs.zip and look at the jsse2Docs/api/index.html file.

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, provided that the debug version of JSSE is installed.

Documentation

See the ″Security: Resources for learning″ 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

There is more information documented for the JCE APIs in the `http://www.ibm.com/developerworks/java/jdk/security/jceDocs.zip` file.

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″ 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. J2EE security guards access to Web resources such as servlets, JavaServer Pages (JSP) files and 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**.

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 tracing and logging″ in the information center 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″ in the *Securing applications* PDF.

```
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;
};
```

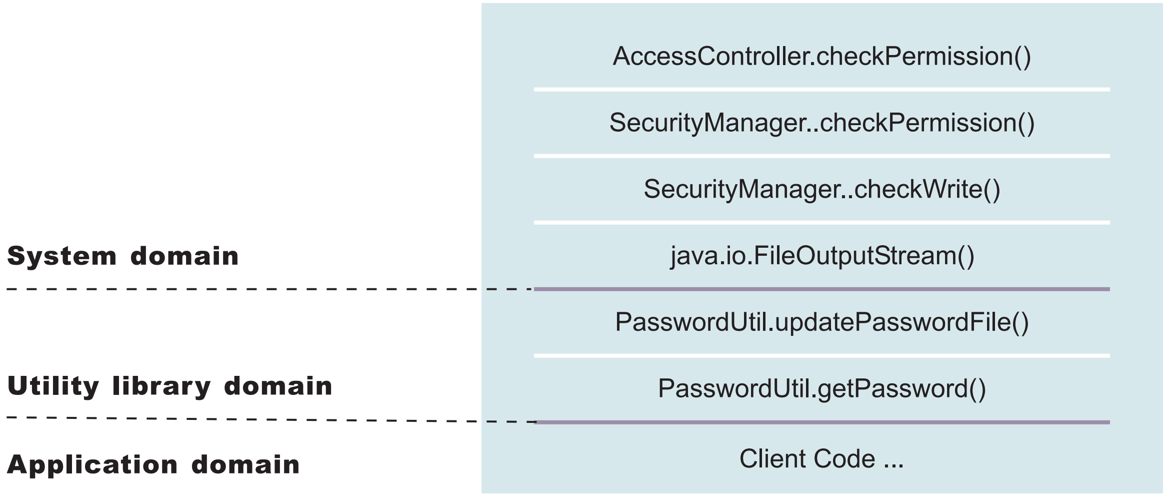
The algorithm requires that all the classes or callers on the call stack have the permissions when a java.security.AccessController.checkPermission method is performed or the request is denied and a java.security.AccessControlException exception is created. However, if the caller is marked as *privileged* 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 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.


AccessController.checkPermission()
SecurityManager..checkPermission()
SecurityManager..checkWrite()
java.io.FileOutputStream()
PasswordUtil.updatePasswordFile()
PasswordUtil.getPassword()
Client Code ...
System domain
Utility library domain
Application domain

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 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 APIs. 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″ article in the information center.
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 1734.

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 1747 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).

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 components troubleshooting tips" in the information center 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.

1. Start policytool. Enter `%{`*`was.install.root`*`}/java/jre/bin/policytool` from a command prompt.

   The policytool window opens. The policytool looks for the `.java.policy` file in your home directory. If it does not exist, an Error message displays. Click **OK**.
2. Click **File** > **Open**.
3. 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.
4. 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`.
5. 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**.
6. 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`.

7. Click **OK** to close the Permissions window. Modified permission entries of the specified code base display.
8. Click **Done** to close the window. Modified code base entries are listed. Repeat steps 4 through 8 until you complete editing.
9. Click **File** > **Save** after you finish editing the file.

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 J2EE 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.

Policy file	Location
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

Symbol	Meaning
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 WebSphere Application Server, Version 5.1 IBM Developer Kit, Java Technology Edition Version 1.4.x. 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.x 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 1743 article for the list of available policy files supported by WebSphere Application Server Version.

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.
   - If the permission is required by all of the WebSphere Application Server enterprise applications in the node, refer to Configuring spi.policy files.

- 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″ in the information center. Otherwise, update the `was.policy` file. Refer to ″Configuring was.policy files″ and ″Adding the was.policy file to applications″ in the information centere .
- If the permission is required by shared libraries, refer to Configuring library.policy files.
- If the permission is required by SPI libraries, refer to Configuring spi.policy files.

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.

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 AccessControlException.

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.

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.

Modify the app.policy file with the Policy Tool. Changes to the app.policy file are local for the node.

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.

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 could call the System.exit() API and cause the entire WebSphere Application Server process to terminate.
- **setSecurityManager** -- An application could install its own SecurityManager that could either grant more permissions or bypass the default policy the WebSphere Application Server SecurityManager enforces.

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 1743 for the list of available policy files supported by WebSphere Application Server Version 5. 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

Symbol	Definition
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 1742.
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 1757.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″ in the information center.
   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 1742.

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 1739.

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 1743 for the list of available policy files supported by WebSphere Application Server Version 6.

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.

Modify the `spi.policy` file with the Policy Tool.

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 1743 for the list of available policy files supported by WebSphere Application Server Version 5. 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.

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 1739.

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 1742
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″ in the information center.
   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 1743 article for the list of available policy files supported by WebSphere Application Server Version 5.

There are two types of policy files supported by WebSphere Application Server Version 5, 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.
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 1747.
   - If the permission is required only by applets and client containers, update the `client.policy` file. Refer to "Configuring client.policy files" on page 1763.
   - 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 1762.
   - 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 1739.

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 1743 for the list of available policy files supported by WebSphere

Application Server Version 5.x or later. 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 1742. 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 1739.

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 1743 for the list of available policy files supported by WebSphere Application Server Version 5. 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 1742 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 1739.

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 1743 for the list of available policy files supported by WebSphere Application Server Version 5. 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" in the information center 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 1739.

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

Starting from Version 3.x, WebSphere Application Server installed a 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 APIs 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.

However, Java 2 security was not a fully supported feature prior to Version 5. 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.

Partial support was introduced since the version 4.02 product release. Prior to version 4.0.2, Java 2 security was not supported. From version 4.02 and later, 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 version 5.x and later, which means all permissions are enforced. The default Java 2 security policy for enterprise application is the recommended permission set defined by the J2EE 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: Enterprise applications that run on Version 4.0.x with Java 2 security enabled are not guaranteed to run successfully when migrating to Version 5 (when Java 2 security is enabled), even if the Java 2 security policy is migrated properly. The default Java 2 security policy for enterprise applications is much more stringent and all permissions are enforced in Version 5. 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.

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 Version 5. 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 from Version 4.0.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 1743 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 1753.
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 1739.

   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.

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 the ″Developing secured applications″ and ″Assembling secured applications″ articles in the information center. 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 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) required prior to the step titled **Map security roles to users and groups**.
2. Assign users and groups to roles. For more information, see “Assigning users and groups to roles.”
3. Map users to RunAs roles if RunAs roles exist in the application. For more information, see “Assigning users to RunAs roles” on page 1775.
4. 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.
5. 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

Before you perform this task:

- Secure the Web applications and EJB applications where new roles were created and assigned to Web and 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.

Since the default active registry is LocalOS, it is not necessary, although it is recommended, that you enable security if you want to use the LocalOS 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 at the bottom.

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** check box 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 1775 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 mappings:

Use this page to map security roles to users. You can map roles to specific users, to specific groups, or to different categories.

To view this administrative console page, click **Application** > **Install New Application**. While running the Application Installation Wizard, prompts appear to help you map security roles to users or groups. To change role to user or group mappings for deployed applications, click **Application** > **Enterprise Application** > *deployed_application* > **Map security roles to users/groups**.

Users:

Specifies the users for role mapping. Verify that the users are defined in your chosen user registry.

To change the roles to users mapping, click **Manage Application >***application* **> Map security roles to users**.

Data type:	String

Groups:

Specifies the groups for role mapping. Verify that the groups are defined in your chosen user registry.

To change the roles to users mapping, click **Manage Application >***application* **> Map security roles to groups**.

Data type:	String

Roles:

Specifies the roles to which you want to map users and groups. Role privileges give users and groups permission to run as specified.

Select the check boxes to choose a role or a set of roles. Click **Look-up Users** to map users to the roles that you have selected. Click **Look-up Groups** to map groups to the selected roles. Use the check boxes to map roles to **EVERYONE** or **ALL AUTHENTICATED** special subject.

Data type:	String

Everyone:

Specifies to map roles to everyone. Mapping a role to everyone means that anyone can access resources protected by this role, and essentially, there is no security.

Data type:	Boolean

All Authenticated:

Specifies to authenticate all users. Roles are mapped to all authenticated users, and all authenticated users in the selected user registry are granted access to the role.

Data type:	Boolean

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**--least privileged that basically allows a user to view the server configuration and current state
- **Configurator**--monitor privilege plus the ability to change the server configuration
- **Operator**--monitor privilege plus the ability to change the run time state, such as starting or stopping services
- **Administrator**--operator plus configurator privilege

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:

Look up users and groups settings:

Use this page to select users and groups for security roles.

To view this administrative console page, click **Applications** > **Enterprise Applications** > *application_name* > **Map security roles to users/groups** > **Look up users or groups** button.

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. Use the Search field to display users in the Available Users list. Click the arrows to add users from the Available Users list to the Selected Users list.

Limit:

Specifies the maximum number of users/groups that can be returned when assigning users/groups to roles.

A value of 0 implies a return of all users/groups that match the pattern. You can either increase the limit or refine the search pattern to get all the entries.

Data type	Integer
Units	User name
Default	20
Range	0 or more

Pattern:

Indicates the search pattern used to search for the entries.

The pattern field should contain the search pattern that should be used to search for the entries. For example, bob* will search all users or groups starting with bob. A limit of 0 gets all the entries that match the pattern and should be used only when a small number users/groups match that pattern in the registry. If the registry contains more entries that match the pattern than requested for, a message shows in the console to indicate that there are more entries in the registry.

Data type	String
Units	Number of users
Default	20
Range	A-Z with *

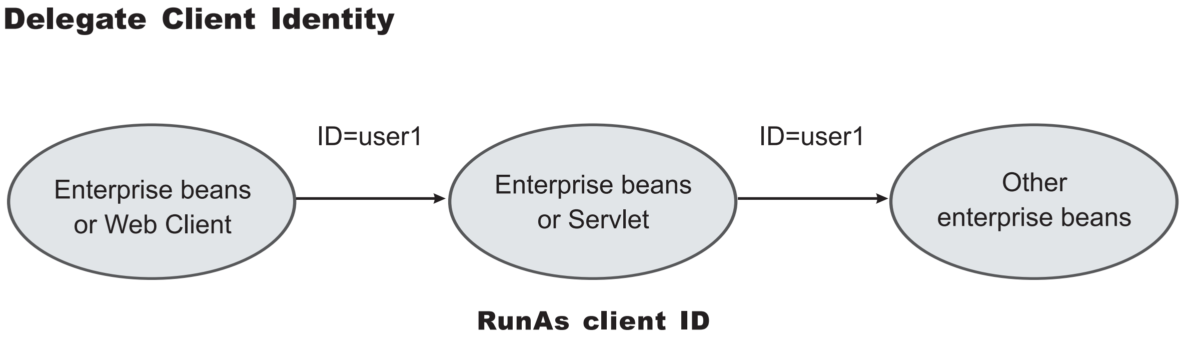
Delegations

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.

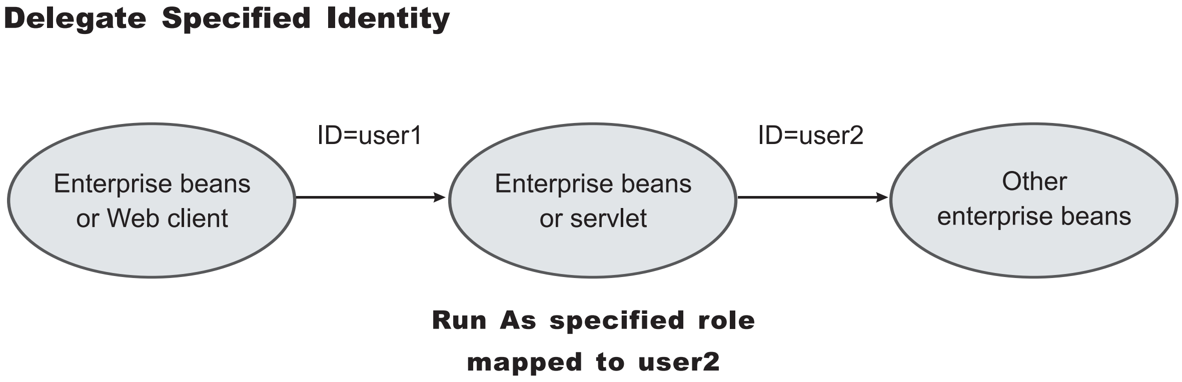
The IBM extension supports Enterprise JavaBeans (EJB) to propagate to the server ID when invoking other entity beans. There are three types of delegations:

- Delegate (RunAs) Client Identity
- Delegate (RunAs) Specified Identity
- Delegate (RunAs) System Identity

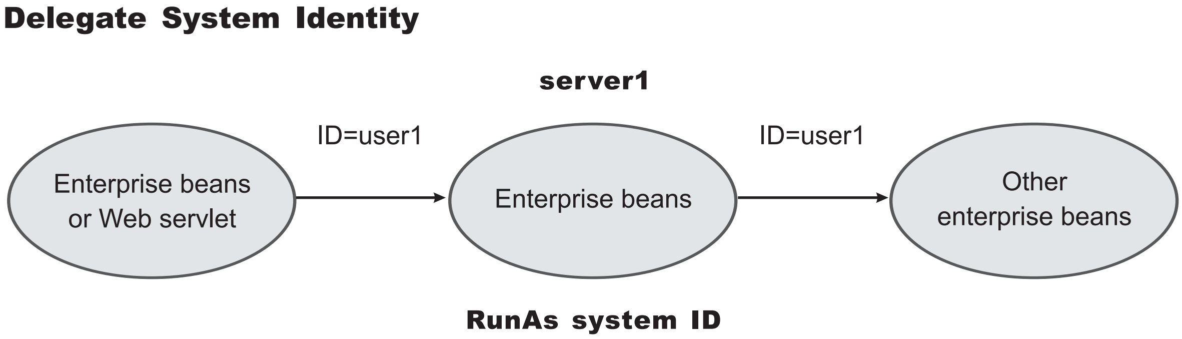
Delegate (RunAs) Client Identity



Delegate (RunAs) Specified Identity


Delegate Specified Identity
ID=user1
ID=user2
Enterprise beans or Web client
Enterprise beans or servlet
Other enterprise beans
Run As specified role mapped to user2

Delegate (RunAs) System Identity

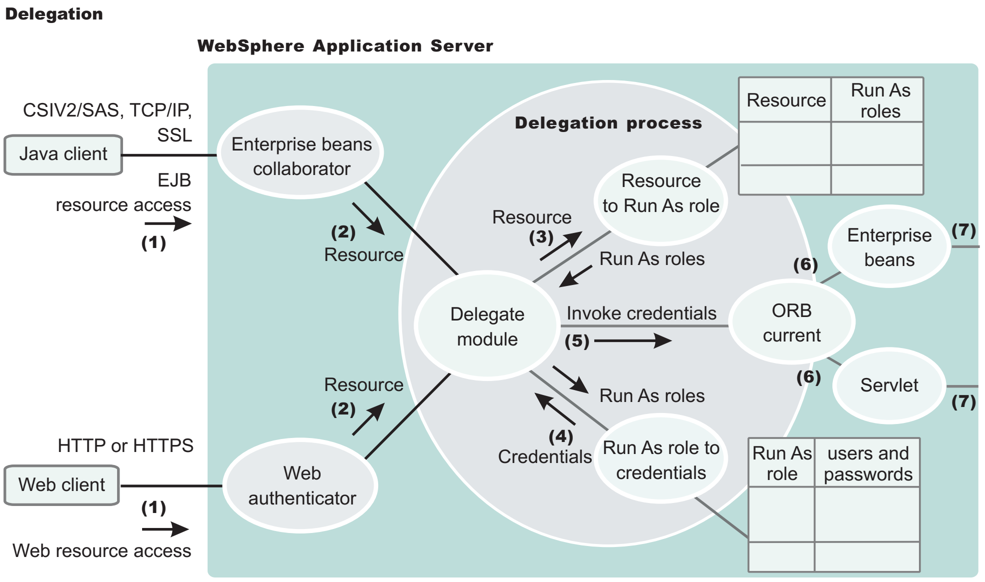


The EJB specification only supports delegation (RunAs) at the EJB level. But an IBM extension allows EJB method level RunAs specification. Method EJB method level runAs specification allows one to specify a different RunAs role for different methods within the same enterprise beans.

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.The resulting credential is stored in the ORB Current as an invocation credential (5). Servlet and enterprise beans when invoking other enterprise beans pick up the invocation credential from the ORB Current (6) and call the next enterprise beans (7).

Assigning users to RunAs roles

Before you perform this task,

- 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 1768 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 1768 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. role1, if user1 or a group that user1 belongs to, already is assigned to role1. 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.

Unprotected EJB 2.0 methods protection settings:

Use this page to verify that unprotected EJB 2.0 methods have the correct level of protection before you map users to roles.

To view this administrative console page, click **Application** > **Install New Application**. While running the Install New Application Wizard, prompts appear to help you map security roles to users.

Exclude:

Specifies that the method is completely protected.

Data type:	Check box
Default:	Cleared

Uncheck:

Specifies that everyone can access the security method.

Data type:	Check box
Default:	Uncheck

Specify role:

Specifies the EJB level of protection based on the security role.

The roles listed in this menu are obtained from the application scope. If the selected role is not in the module, then it is added to the modules or Java archive (JAR) files.

Data type:	String
Units:	Role

Module name:

Specifies the name of the module.

If a module name appears in this list, then the module contains unprotected EJB methods.

Data type:	String
Units:	Module name

Protection:

Specifies the level of protection assigned to a particular module name.

Data type:	String
Default:	Cleared

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**--least privileged that basically allows a user to view the WebSphere configuration and current state
- **Configurator**--monitor privilege plus the ability to change the WebSphere configuration
- **Operator**--monitor privilege plus the ability to change runtime state, such as starting or stopping services for example
- **Administrator**--operator plus configurator privilege

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 1768.

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 1775.
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″ in the information center.
6. Use an assembly tool to modify roles, method permissions, auth-constraints, data-constraints and so on. For more information, see ″Assembling applications″ in the information center.
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. See ″Assembling applications″ in the information center. 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 1783.
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 1783. For more information on configured name bindings, refer to “Configured name bindings” on page 1786.

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. For details, refer to the *Troubleshooting and support* PDF.

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.

Logical View of a Cell's Name Space

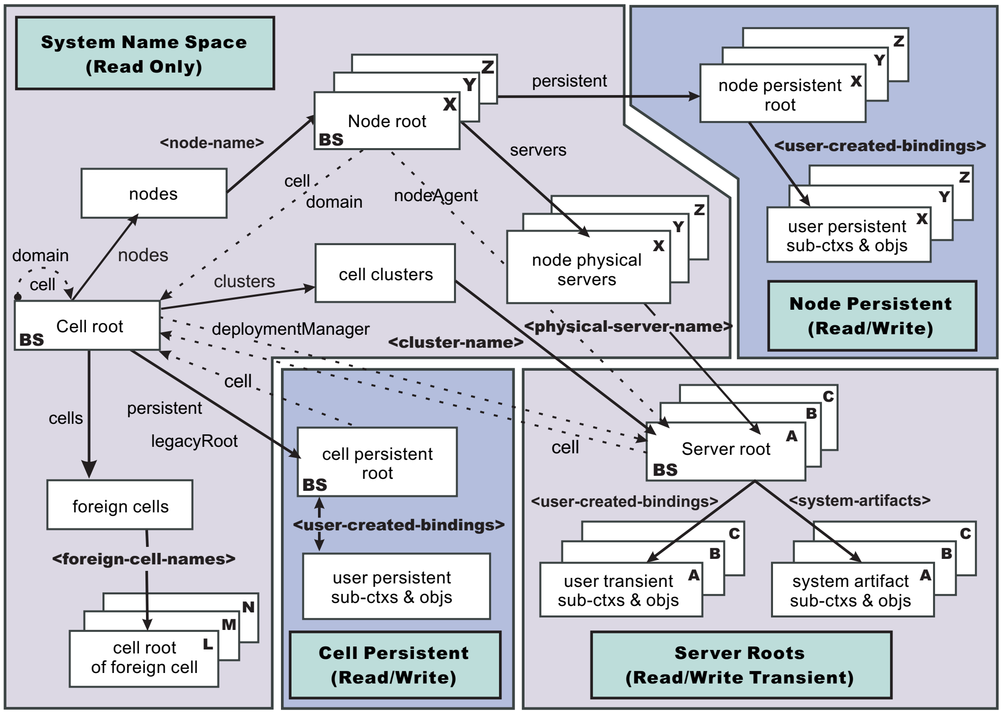


Figure 8. Name Space Logical View

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.

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.

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.

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 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 .

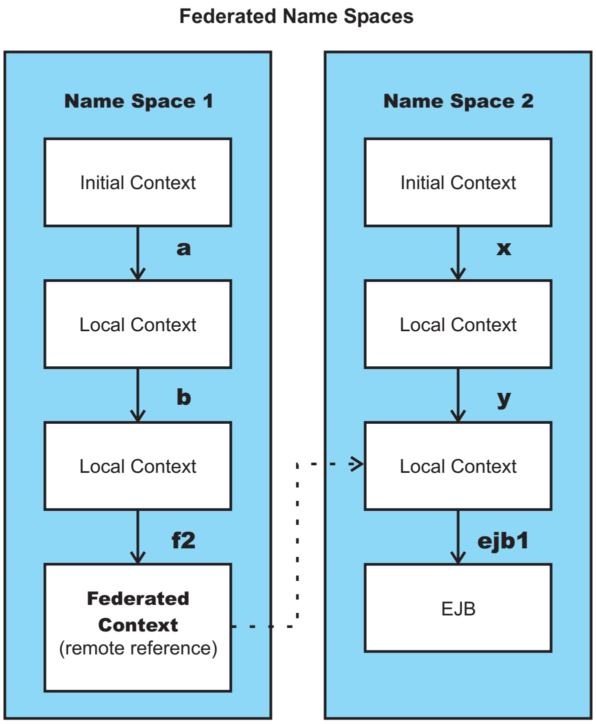
String: String constant

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 the information center 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^{TM} 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 **Servers > Application Servers >** ***server_name*** **> Administration > Server Components > Name 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 should only contain the bootstrap addresses of members of the same cluster. Otherwise, incorrect behavior may occur.

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 an 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
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 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.CORBA.iiop.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);
...
```

```
...
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” on page 1816 for more information. Before making these adjustments, read “Object Request Broker tuning guidelines” on page 1815.
- Adjust thread-pool settings used by the ORB for handling Internet InterORB Protocol (IIOP) connections. See ″Thread pool settings″ in the information center.
- If problems with the ORB arise, see ″Object request broker component troubleshooting tips″ in the information center.

  For help in troubleshooting, see “Object Request Broker communications trace” on page 1824.

Object Request Brokers

An Object Request Broker (ORB) manages the interaction between clients and servers, using the Internet InterORB Protocol (IIOP). It enables clients to make requests and receive responses from servers in a network-distributed environment.

The ORB provides a framework for clients to locate objects in the network and to call operations on those objects as if the remote objects are located in the same running process as the client, providing location transparency. The client calls an operation on a local object, known as a *stub*. The stub forwards the request to the remote object, where the operation runs and the results are returned to the client.

The client-side ORB is responsible for creating an IIOP request that contains the operation and required parameters, and for sending the request on the network. The server-side ORB receives the IIOP request, locates the target object, invokes the requested operation, and returns the results to the client. The client-side ORB demarshals the returned results and passes the result to the stub, which, in turn, returns to the client application, as if the operation had been run locally.

This product uses an ORB to manage communication between client applications and server applications as well as communication among product components. During product installation, default property values are set when the ORB is initialized. These properties control the run-time behavior of the ORB and can also affect the behavior of product components that are tightly integrated with the ORB, such as security. This product does not support the use of multiple ORB instances.

Logical pool distribution

The Logical pool distribution (LPD) thread pool mechanism implements a strategy for improving the performance of requests that have shorter run times. Do not configure LPD unless you have already configured it in a previous release of WebSphere Application Server.

The need for LPD is indicated by a mixture of Enterprise JavaBeans (EJB) requests where the run times vary across the request types, and the ORB thread pool must be constrained for performance reasons. In this case, longer run time requests might tend to prolong the response times for shorter requests by denying them adequate access to threads in the thread pool. LPD provides a mechanism that allows shorter requests greater access to the threads.

LPD divides the Object Request Broker (ORB) thread pool into logical pools, as configured by the administrator using ORB custom properties starting that start with the following:

```
com.ibm.websphere.threadpool.strategy.*
```

The size of each pool is a percentage of the maximum number of ORB threads. The sum of the logical pool percentages must equal 100.

When LPD is active, incoming ORB requests are vectored, or pointed, to a pool based on historical run time history for the request type. The request type is determined by the method, which is qualified internally as unique across components. The LPD mechanism adjusts pool targets at runtime to optimize the distribution of requests across logical pools.

The LPD mechanism can be tuned after it is enabled. Response time, throughput measurements, and statistics produced by the LPD mechanism drive the tuning process.

Object Request Broker tuning guidelines

The following options exist for improving the performance of the Object Request Broker (ORB). Tuning results vary among systems and applications.

- **Logical pool distribution (LPD) mechanism**

  Do not configure LPD unless you have already configured it with a previous version of WebSphere Application Server.

  If you suspect that requests with longer completion times are elongating the response times for shorter completion time requests by denying them adequate access to threads in the thread pool, LPD provides a mechanism to allow the shorter requests greater access to execution threads.

  For more information, see “Logical pool distribution” on page 1814
- **ORB timeout**

  If Web clients that access Java applications running in the product environment are consistently experiencing problems with their requests, and the problem cannot be traced to other sources and addressed through other solutions, consider setting an ORB timeout value and adjusting it for your environment. A list of timeout scenarios follows:
  – Web browsers vary in their language for indicating that they have timed out. Usually, the problem is announced as a connection failure or a no-path-to-server message.
  – Set an ORB timeout value to less than the time after which a Web client eventually times out. Because it can be difficult to tell how long Web clients wait before timing out, setting an ORB timeout value requires experimentation. The ideal testing environment features some simulated network failures for testing the proposed setting value.
  – Empirical results from limited testing indicate that 30 seconds is a reasonable starting value. Ensure that this setting is not too low. To fine tune the setting, find a value that is not too low. Gradually decrease the setting until reaching the threshhold at which the value becomes too low. Set the value a little higher than the threshold.
  – When an ORB timeout value is set too low, the symptom is numerous CORBA ’NO_RESPONSE’ exceptions, which occur even for some valid requests. The value is likely to be too low if requests that should have been successful, for example, the server is not down, are being lost or refused.

  Timeout adjustments: Do not adjust an ORB timeout value unless you have a problem. Configuring a value that is inappropriate for the environment can create a problem. An incorrect value can produce results worse than the original problem.

  You can adjust timeout intervals for the product Java ORB through the following administrative settings:
  – **Request timeout**, the number of seconds to wait before timing out on most pending ORB requests if the network fails
  – **Locate request timeout**, the number of seconds to wait before timing out on a locate-request message
- **com.ibm.CORBA.ConnectTimeout system property**

  The `com.ibm.CORBA.ConnectTimeout` property specifies the maximum time in seconds that the client ORB waits before timing out when establishing a new socket connection with a remote server ORB. This property is intended for use by client applications. It is not used by the application server.

  You can specify the com.ibm.CORBA.ConnectTimeout property in the `orb.properties` file, or you can add the property when running the `launchclient` script. This example specifies a maximum timeout value of ten seconds:

  ```
  launchclient clientapp.ear -CCDcom.ibm.com.CORBA.ConnectTimeout=10...
  ```

  You can 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 may result in exceptions if the remote server does not have enough time to complete the connection.

Valid Range	0-300 (seconds)
Default	0 (the client ORB waits indefinitely)

- **com.ibm.CORBA.numJNIReaders system property**

  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 be shared for request handling when the native selector mechanism is enabled.

Valid Range	0-2147483647
Default	2

- **Determining the ORB message size**

  The ORB separates messages into fragments to send over the ORB connection. You can configure this fragment size through the com.ibm.CORBA.FragmentSize parameter.

  To determine the size of the messages that transfer over the ORB and the number of required fragments:

  1. In the administrative console, enable ORB tracing in the ORB Properties page. See “Object Request Broker service settings” for more information.
  2. Enable ORBRas tracing from the logging and tracing page.
  3. Increase the trace file sizes because tracing can generate a lot of data.
  4. Restart the server and run at least one iteration (preferably several) of the case that you are measuring.
  5. Look at the traceable file and do a search for `Fragment to follow: Yes`.

     This indicates that the ORB transmitted a fragment, but it still has at least one remaining fragment to send before the entire message has arrived. A `Fragment to follow: No` value indicates that the particular fragment is the last in the entire message. This fragment can also be the first, if the message fit entirely into one fragment.

     If you go to the spot where `Fragment to follow: Yes` is located, you find a block that looks similar to the following example:

Fragment to follow:	Yes
Message size:	4988 (0x137C)
--	
Request ID:	1411

This example indicates that the amount of data in the fragment is 4988 bytes and the Request ID is 1411. If you search for all occurrences of `Request ID: 1411`, you can see the number of fragments used to send that particular message. If you add all the associated message sizes, you have the total size of the message that is being sent through the ORB.

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.

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.

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 pass by reference, the parameters are copied to the stack before every remote method call is made, which can be expensive.

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 Pass by reference can improve performance up to 50%. Pass by reference helps performance only where non-primitive object types are passed as parameters. Therefore, int and floats are always copied, regardless of the call model.

Enable this property with caution, because unexpected behavior can occur. If an object reference is modified by the remote method, the caller might change.

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 Pass by reference is enabled. Setting Pass by reference 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 Pass by reference 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.

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 (shown on 2 lines for publication):

```
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 1827.

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 1827.

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” on page 1814.

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.

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.

For incremental cell upgrade instructions, see ″Migrating a V5.x managed node to a V6 managed node″ and ″Migrating Network Deployment, V5.x to a V6 deployment manager″ in the information center.

Object Request Broker communications trace

The Object Request Broker (ORB) communications trace, typically referred to as *CommTrace*, contains the sequence of General InterORB Protocol (GIOP) messages sent and received by the ORB when the application is running. It might be necessary to understand the low-level sequence of client-to-server or server-to-server interactions during problem determination. This topic uses trace entries from log examples to explain the contents of the log and help you understand the interaction sequence. It focuses only in the GIOP messages and does not discuss in detail additional trace information that displays when intervening with the GIOP-message boundaries.

Location

When ORB tracing is enabled, this information is placed in the `*install_root*/logs/trace` directory.

About the ORB trace file

The following are properties of the file that is created when ORB tracing is enabled.
- Read-only
- Updated by the administrative function
- Use this file to localize and resolve ORB-related problems.

How to interpret the output

The following sections refer to sample log output found later in this topic.

Identifying information

The start of a GIOP message is identified by a line that contains either OUT GOING: or IN COMING: depending on whether the message is sent or received by the process.

Following the identifying line entry is a series of items, formatted for convenience, with information extracted from the raw message that identifies the endpoints in this particular message interaction. See lines 3-13 in both examples. The formatted items include the following:
- GIOP message type (line 3)
- Date and time that the message was recorded (line 4)
- Information that is useful to identify the thread that is running when the message records, and other thread-specific information (line 5)
- Local and remote TCP/IP ports used for the interaction (lines 6 through 9)
- GIOP version, byte order, an indication of whether the message is a fragment, and message size (lines 10 through 13)

Request ID, response expected and reply status

Following the introductory message information, the request ID is an integer generated by the ORB. It is used to identify and associate each request with its corresponding reply. This association is necessary because the ORB can receive requests from multiple clients and must be able to associate each reply with the corresponding originating request.

- Lines 15-17 in the request example show the request ID, followed by an indication to the receiving endpoint that a response is expected (CORBA supports sending one-way requests for which a response is not expected.)
- Line 15 in the reply example shows the request ID; line 33 shows the reply status received after completing the previously sent request.

Object Key

Lines 18-20 in the request example show the object key, the internal representation used by the ORB to identify and locate the target object intended to receive the request message. Object keys are not standardized.

Operation

Line 21 in the request example shows the name of the operation that the target object starts in the receiving endpoint. In this example, the specific operation requested is named `_get_value`.

Service context information

The service contexts in the message are also formatted for convenience. Each GIOP message might contain a sequence of service contexts sent and received by each endpoint. Service contexts, identified uniquely with an ID, contain data used in the specific interaction, such as security, character code set conversion, and ORB version information. The content of some of the service contexts is standardized and specified by OMG, while other service contexts are proprietary and specified by each vendor. IBM-specific service contexts are identified with IDs that begin with `0x4942`.

Lines 22-41 in the request example illustrate typical service context entries. Three service contexts are in the request message, as shown in line 22. The ID, length of data, and raw data for each service context is printed next. Lines 23-25 show an IBM-proprietary context, as indicated by the `0x49424D12` ID. Lines 26-41 show two standard service contexts, identified by `0x6` ID (line 26) and the `0x1` ID (line 39).

Lines 16-32 in the reply example illustrate two service contexts, one IBM-proprietary (line 17) and one standardized (line 20).

For the definition of the standardized service contexts, see the CORBA specification. Service context 0x1 (CORBA::IOP::CodeSets) is used to publish the character code sets supported by the ORB in order to negotiate and determine the code set used to transmit character data. Service context 0x6 (CORBA::IOP::SendingContextRunTime) is used by Remote Method Invocation over the Internet Inter-ORB Protocol (RMI-IIOP) to provide the receiving endpoint with the IOR for the SendingContextRuntime object. IBM service context 0x49424D12 is used to publish ORB PartnerVersion information to support release-to-release interoperability between sending and receiving ORBs.

Data offset

Line 42 in the request example shows the offset, relative to the beginning of the GIOP message, where the remainder body of the request or reply message is located. This portion of the message is specific to each operation and varies from operation to operation. Therefore, it is not formatted, as the specific contents are not known by the ORB. The offset is printed as an aid to quickly locating the operation-specific data in the raw GIOP message dump, which follows the data offset.

Raw GIOP message dump

Starting at line 45 in the request example and line 36 in the reply example, a raw dump of the entire GIOP message is printed in hexadecimal format. Request messages contain the parameters required by the given operation and reply messages contain the return values and content of output parameters as required by the given operation. For brevity, not all of the raw data is in the figures.

Sample Log Entry - GIOP Request

```
1.  OUT GOING:

3.  Request Message
4.  Date:         April 17, 2002 10:00:43 PM CDT
5.  Thread Info:  P=842115:O=1:CT
6.  Local Port:   1243 (0x4DB)
```

```
7.  Local IP:      jdoe.austin.ibm.com/192.168.1.101
8.  Remote Port:   1242 (0x4DA)
9.  Remote IP:     jdoe.austin.ibm.com/192.168.1.101
10.  GIOP Version:  1.2
11. Byte order:    big endian
12. Fragment to follow: No
13. Message size:  268 (0x10C)
--
15. Request ID:        5
16. Response Flag:     WITH_TARGET
17. Target Address:     0
18. Object Key:        length = 24 (0x18)
                       4B4D4249 00000010 BA4D6D34 000E0008
                       00000000 00000000
21. Operation:         _get_value
22. Service Context:   length = 3 (0x3)
23. Context ID:  1229081874 (0x49424D12)
24. Context data:  length = 8 (0x8)
                       00000000 13100003
26. Context ID:  6 (0x6)
27. Context data:  length = 164 (0xA4)
                       00000000 00000028 49444C3A 6F6D672E
                       6F72672F 53656E64 696E6743 6F6E7465
                       78742F43 6F646542 6173653A 312E3000
                       00000001 00000000 00000068 00010200
                       0000000E 3139322E 3136382E 312E3130
                       310004DC 00000018 4B4D4249 00000010
                       BA4D6D69 000E0008 00000000 00000000
                       00000002 00000001 00000018 00000000
                       00010001 00000001 00010020 00010100
                       00000000 49424D0A 00000008 00000000
                       13100003
39. Context ID:  1 (0x1)
40. Context data:  length = 12 (0xC)
                       00000000 00010001 00010100
42. Data Offset:       118


45. 0000: 47494F50 01020000 0000010C 00000005   GIOP............
46. 0010: 03000000 00000000 00000018 4B4D4249   ............KMBI
47. 0020: [remainder of message body deleted for brevity]
```

Sample Log Entry - GIOP Reply

```
1.  IN COMING:

3.  Reply Message
4.  Date:          April 17, 2002 10:00:47 PM CDT
5.  Thread Info:   RT=0:P=842115:O=1:com.ibm.rmi.transport.TCPTransportConnection
5a (line 5 broken for publication).   remoteHost=192.168.1.101 remotePort=1242 localPort=1243
6.  Local Port:    1243 (0x4DB)
7.  Local IP:      jdoe.austin.ibm.com/192.168.1.101
8.  Remote Port:   1242 (0x4DA)
9.  Remote IP:     jdoe.austin.ibm.com/192.168.1.101
10. GIOP Version:  1.2
11. Byte order:    big endian
12. Fragment to follow: No
13. Message size:  208 (0xD0)
--
15. Request ID:        5
16. Service Context:   length = 2 (0x2)
17. Context ID:  1229081874 (0x49424D12)
18. Context data:  length = 8 (0x8)
                       00000000 13100003
20. Context ID:  6 (0x6)
21. Context data:  length = 164 (0xA4)
                       00000000 00000028 49444C3A 6F6D672E
```

```
                     6F72672F 53656E64 696E6743 6F6E7465
                     78742F43 6F646542 6173653A 312E3000
                     00000001 00000000 00000068 00010200
                     0000000E 3139322E 3136382E 312E3130
                     310004DA 00000018 4B4D4249 00000010
                     BA4D6D34 000E0008 00000001 00000000
                     00000002 00000001 00000018 00000000
                     00010001 00000001 00010020 00010100
                     00000000 49424D0A 00000008 00000000
                     13100003
33. Reply Status:    NO_EXCEPTION


36. 0000: 47494F50 01020001 000000D0 00000005   GIOP...........
37. 0010: 00000000 00000002 49424D12 00000008   ........IBM.....
38. 0020: [remainder of message body deleted for brevity]
```

Client-side programming tips for the Java Object Request Broker service

This topic includes programming tips for applications that communicate with the client-side Object Request Broker (ORB) that is part of the Java ORB service.

Resolution of initial references to services

Client applications can use the ORBInitRef and ORBDefaultInitRef properties to configure the network location that the Java ORB service uses to find a service such as naming. When set, these properties are included in the parameters that are used to initialize the ORB, as illustrated in the following example:

```
org.omg.CORBA.ORB.init(java.lang.String[] args,
                       java.util.Properties props)
```

You can set these properties in client code or by command-line argument. It is possible to specify more than one service location by using multiple ORBInitRef property settings (one for each service), but only a single ORBDefaultInitRef value can be specified. For 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 1830.

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`.

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	

Java name	OSF registry ID	Comments
Cp1255	0x100204E7	
Cp1256	0x100204E8	
Cp1257	0x100204E9	
Cp943C	0x100203AF	
Cp943	0x100203AF	
Cp949C	0x100203B5	
Cp949	0x100203B5	
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" on page 1831
- "Administration" on page 1831
- "Programming specifications" on page 1831

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.

Transactions

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:** If you want to change the directory in which transaction logs are written, type the full pathname of the directory in the **Transaction log directory** field. You can check the current runtime value of **Transaction log directory**, by clicking the Runtime tab.

   When using WebSphere Application Server without High Availability support, you can leave the recovery log configuration for persistent services (such as the transactions service) unset. The application server assumes a default location within the appropriate profile directory. When High Availability support is enabled, this default may not be visible from all servers in the cluster (for example, if they are in different profiles or physical nodes.) As a result, it is recommended that the recovery log location be configured for each server in the cluster before enabling High Availability.

You can also specify a size for the transaction logs, as described in the following step.

Note: If you change the transaction log directory, you should apply the change and restart the application server as soon as possible, to minimize the risk of problems caused that might occur before the application server is restarted. For example, if a problem causes the server to fail (with in-flight transactions), the server next starts with the new log directory and is unable to automatically resolve in-flight transactions that were recorded in the old log directory.

6. **Optional:** If you want to change the default file size of transaction log files, modify the **Transaction log directory** field to include a file size setting, in the following format:

   directory_name;*file_size*

   Where
   - *directory_name* is the name of the transaction log directory
   - *file_size* is the new default size specified in bytes. The *n*K or *n*M suffix can be used to indicate kilobytes or megabytes. If you do not specify a file size value, the default value of 1M is used.

   For example, `c:\tranlogs;2M` indicates the files are to be created with 2M bytes size and stored in the directory c:\tranlogs.

   In a non-production environment, you can use the transaction log directory value of `;0` to disable transaction logging. (There must be no directory name element before the size element of 0.) You should not disable transaction logging in a production environment, because this prevents recovery after a system failure and, therefore, data integrity cannot be guaranteed.

7. **Optional:** Review or change the value of transaction timeout properties:

   Total transaction lifetime timeout
   Type the number of seconds a transaction can remain inactive before it is ended by the transaction service. A value of 0 (zero) indicates that there is no timeout limit.

   Maximum transaction timeout
   Type the number of seconds for which a transaction started by or propagated into this application server can run before it is ended by the transaction service.

   A value of 0 (zero) indicates that there is no timeout limit.

   Client inactivity timeout
   Type the number of seconds after which a client is considered inactive and the transaction service ends any transactions associated with that client. A value of 0 (zero) indicates that there is no timeout limit.

8. **Optional:** Review or change heuristic-related properties:

   Heuristic retry limit
   The number of times that the application server retries a completion signal, such as commit or rollback, after a transient exception from a resource manager or remote partner.

   Heuristic retry wait
   The number of seconds that the application server waits before retrying a completion signal, such as commit or rollback, after a transient exception from a resource manager or remote partner.

   Enable logging for heuristic reporting
   Select this property to enable the application server to log ″about to commit one-phase resource″ events from transactions that involve a one-phase commit resource and two-phase commit resources.

   Heuristic completion direction
   Select the direction used to complete a transaction that has a heuristic outcome; either the application server commits or rolls back the transaction, or depends on manual completion by the administrator.

9. Review or change other configuration properties, to suit your requirements. For more information about the properties of the transaction service, see "Transaction service settings."
10. Click **OK** and save.
11. Stop then restart the application server.

    If you change the transaction log directory configuration property to an incorrect directory name, the application server will restart but be unable to open the transaction logs. You should change the configuration property to a valid directory name, then restart the application server.

    If you are running the application server as non-root, modify the permissions on the new transaction log location. If you want to use peer recovery of transactions on a shared device with non-root users, make sure that your non-root users and groups have matching identification numbers across machines

Transaction service settings

Use this page to 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 name of a directory for this server where the transaction service stores log files for recovery.

A blank value in the server configuration is expanded by the transaction service at startup as the directory *install_root*/tranlog/*cell_name*/*node_name*/*server_name*.

When the application running on WebSphere Application Server accesses more than one resource, Application Server stores transaction information to properly coordinate and manage the distributed transaction. In a higher transaction load, this persistence slows down performance of the application server due to its dependency on the operating system and the underlying storage systems.

To achieve better performance, move the transaction log files to a storage device with more physical disk drives, or preferably RAID storage array disk drives. When the log files are moved to the file systems on such disks, the task of writing data to the physical media is shared across the multiple disk drives. This allows more concurrent access to persist transaction information and faster access to that data from the logs.

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.

Consider setting this property when the application server shows one or more of following signs:
- CPU utilization remains low despite an increase in transactions
- Transactions fail with several timeouts
- Transaction rollbacks occur with *unable to enlist transaction* exception
- Application server hangs in middle of a run and requires the server to be restarted
- The disk on which an application server is running shows higher utilization

Data type	String
Default	Initial value is the *install_root*/tranlog/*cell_name*/*node_name*/*server_name* directory and a default size of 1MB.

Recommended	Create a file system with at least 3-4 disk drives raided together in a RAID-0 configuration. Then, create the transaction log on this file system with the default size. When the server is running under load, check the disk input and output. If disk input and output time is more then 5%, consider adding more physical disks to lower the value.

If you migrate a WebSphere Application Server Version 5 node to Version 6, the stored location of this configuration property is moved from the server level to the node (server index) level to enable high-availability support for transaction logging. 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.

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 2 147 483 647

Client inactivity timeout:

Specifies the maximum duration, in seconds, between transactional requests from a remote client.

Any period of client inactivity that exceeds this timeout results in the transaction 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 XA 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 A value of 0 means that the application server determines the retry wait; the server doubles the retry wait after every 10 failed retries.

Enable logging for heuristic reporting:

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.

This property enables logging for heuristic reporting. If applications are configured to allow one-phase commit resources to participate in two-phase commit transactions, reporting of heuristic outcomes that occur at application server failure requires extra information to be written to the transaction log. If enabled, one additional log write is performed for any transaction that involves both one- and two-phase commit resources. No additional records are written for transactions that do not involve a one-phase commit resource.

Data type	String
Default	Cleared
Range	**Cleared** The application server does not log ″about to commit one-phase resource″ events from transactions that involve a one-phase commit resource and two-phase commit resources. **Selected** The application server does log ″about to commit one-phase resource″ events from transactions that involve a one-phase commit resource and two-phase commit resources.

Heuristic completion direction:

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	Enum
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. Any resource manager than cannot be contacted is left in an in-doubt (prepared) state that needs to be resolved by the administrator using mechanisms specific to the resource manager.

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.

An administrator can heuristically complete the transactions listed on this page independent of the external transaction manager if, for example, the external transaction manager has become unavailable for an unacceptable period of time. 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.

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.
- 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
- Using local transactions
- Developing a WebSphere application to use transactions
- Configuring transaction properties for an application server
- Managing active transactions
- Managing transaction logging for optimum server availability
- Interoperating transactionally between application servers
- Troubleshooting transactions
- Transaction service exceptions
- UserTransaction interface - methods available
- Coordinating access to 1-PC and 2-PC-capable resources within the same transaction
- Implementing WebSphere enterprise applications that use ActivitySessions

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.

Resource managers that offer transaction support can be categorized into those that support two-phase coordination (by offering an XAResource interface) and those that support only one-phase coordination (for example through a LocalTransaction interface). The WebSphere Application Server transaction support provides coordination, within a transaction, for any number of two-phase capable resource managers. It also enables a single one-phase capable resource manager to be used within a transaction in the absence of any other resource managers, although a WebSphere transaction is not necessary in this case.

Under normal circumstances you cannot mix one-phase commit capable resources and two-phase commit capable resources in the same global transaction, because one-phase commit resources cannot support the prepare phase of two-phase commit. There are some special circumstances where it is possible to include mixed-capability resources in the same global transaction:

- In scenarios where there is only a single one-phase commit resource provider that participates in the transaction and where all the two-phase commit resource-providers that participate in the transaction are used in a read-only fashion. In this case, the two-phase commit resources all vote read-only during the prepare phase of two-phase commit. Because the one-phase commit resource provider is the only provider to actually perform any updates, the one-phase commit resource does not need to be prepared.
- In scenarios where there is only a single one-phase commit resource provider that participates in the transaction with one of more two-phase commit resource providers and where last participant support is enabled. Last participant support enables the use of a single one-phase commit capable resource with any number of two-phase commit capable resources in the same global transaction. For more information about last participant support, see Using one-phase and two-phase commit resources in the same transaction.

The ActivitySession service provides an alternative unit-of-work (UOW) scope to that provided by global transaction contexts. It is a distributed context that can be used to coordinate multiple one-phase resource managers. The WebSphere EJB container and deployment tooling support ActivitySessions as an extension to the J2EE programming model. EJBs can be deployed with lifecycles that are influenced by ActivitySession context, as an alternative to transaction context. An application can then interact with a resource manager for the period of a client-scoped ActivitySession, rather than only the duration of an EJB method, and have the resource manager's local transaction outcome directed by the ActivitySession. For more information about ActivitySessions, see Using the ActivitySession service.

Resource manager local transaction (RMLT):

A resource manager local transaction (RMLT) is a resource manager's view of a local transaction; that is, it represents a unit of recovery on a single connection that is managed by the resource manager.

Resource managers include:

- Enterprise Information Systems that are accessed through a resource adapter, as described in the J2EE Connector Architecture 1.0.
- Relational databases that are accessed through a JDBC datasource.
- JMS queue and topic destinations.

Resource managers offer specific interfaces to enable control of their RMLTs. J2EE connector resource adapters that include support for local transactions provide a LocalTransaction interface to enable applications to request that the resource adapter commit or rollback RMLTs. JDBC datasources provide a Connection interface for the same purpose.

The boundary at which all RMLTs must be complete is defined in WebSphere Application Server by a local transaction containment (LTC).

Global transactions:

If an application uses two or more resources, then an external transaction manager is needed to coordinate the updates to both resource managers in a global tansaction.

Global transaction support is available to web and enterprise bean J2EE components and, with some limitation, to application client components. Enterprise bean components can be subdivided into beans that exploit container-managed transactions (CMT) or bean-managed transactions (BMT).

BMT enterprise beans, application client components, and web components can use the Java Transaction API (JTA) UserTransaction interface to define the demarcation of a global transaction. The UserTransaction interface can be obtained by a JNDI lookup of `java:comp/UserTransaction` or from the SessionContext object using the getUserTransaction method..

The UserTransaction is not available to the following components:
- CMT enterprise beans. Any attempt by such beans to obtain the interface results in an exception in accordance with the EJB specification.

Ensure that programs that perform a JNDI lookup of the UserTransaction interface, use an InitialContext that resolves to a local implementation of the interface. Also ensure that such programs use a JNDI location appropriate for the EJB version.

Before the EJB 1.1 specification, the JNDI location of the UserTransaction interface was not specified. Each EJB container implementor defined it in an implementation-specific manner. Earlier versions of WebSphere Application Server, up to and including Version 3.5.x (without EJB 1.1), bind the UserTransaction interface to a JNDI location of jta/usertransaction. WebSphere Application Server Version 4, and later releases, bind the UserTransaction interface at the location defined by EJB 1.1, which is java:comp/UserTransaction. WebSphere Application Server, from Version 5 no longer provides the jta/usertransaction binding within Web and EJB containers to applications at a J2EE level of 1.3 or later. For example, from EJB 2.0 applications can use only the java:comp/UserTransaction location.

A web component or enterprise bean (CMT or BMT) can get the ExtendedJTATransaction interface through a lookup of `java:comp/websphere/ExtendedJTATransaction`. This interface provides access to the transaction identity and a mechanism to receive notification of transaction completion.

Local transaction containment (LTC):

A local transaction containment (LTC) is used to define the application server behavior in an unspecified transaction context.

(Unspecified transaction context is defined in the Enterprise JavaBeans 2.0 (or later) specification; for example, at http://java.sun.com/products/ejb/2.0.html.)

A LTC is a bounded unit-of-work scope within which zero, one, or more resource manager local transactions (RMLTs) can be accessed. The LTC defines the boundary at which all RMLTs must be complete; any incomplete RMLTs are resolved, according to policy, by the container. An LTC is local to a bean instance; it is not shared across beans even if those beans are managed by the same container. LTCs are started by the container before dispatching a method on a J2EE component (such as an enterprise bean or servlet) whenever the dispatch occurs in the absence of a global transaction context. LTCs are completed by the container depending on the application-configured LTC boundary; for example at the end of the method dispatch. There is no programmatic interface to the LTC support; rather LTCs are managed exclusively by the container and configured by the application deployer through transaction attributes in the application deployment descriptor.

A local transaction containment cannot exist concurrently with a global transaction. If application component dispatch occurs in the absence of a global transaction, the container always establishes an LTC for J2EE components at J2EE 1.3 or later. The only exceptions to this are as follows:

- Where application component dispatch occurs without container interposition; for example, for a stateless session bean `create` or a servlet-initiated thread.
- J2EE 1.2 web components.
- J2EE 1.2 BMT enterprise beans.

A local transaction containment can be scoped to an ActivitySession context that lives longer than the enterprise bean method in which it is started, as described in ActivitySessions and transaction contexts.

Local and global transaction considerations: Applications use resources, such as JDBC data sources or connection factories, that are configured through the Resources view of the WebSphere Application Server Administrative Console. How these resources participate in a global transaction depends on the underlying transaction support of the resource provider. For example, most JDBC providers can provide either XA or non-XA versions of a data source. A non-XA data source can support only resource manager local transactions (RMLTs), but an XA data source can support two-phase commit coordination, as well as local transactions.

If an application uses two or more resource providers that support only RMLTs, then atomicity cannot be assured because of the one-phase nature of these resources. To ensure atomic behavior, the application should use resources that support XA coordination and should access them within a global transaction.

If an application uses only one RMLT, the atomic behavior can be guaranteed by the resource manager, which can be accessed under a local transaction containment context.

An application can also access a single resource manager under a global transaction context, even if that resource manager does not support the XA coordination. An application can do this, because WebSphere Application Server performs an “only resource optimization” and interacts with the resource manager under a RMLT. Within a global transaction context, any attempt to use more than one resource provider that supports only RMLTs causes the global transaction to be rolled back.

At any moment, an instance of an enterprise bean can have work outstanding in either a global transaction context or a local transaction containment context, but never both. An instance of an enterprise bean can change from running under one type of context to the other (in either direction), if all outstanding work in the original context is complete. Any violation of this principle causes an exception to be thrown when the enterprise bean tries to start the new context.

Client support for transactions:

This topic describes the support of application clients for the use of transactions.

Application clients running in a J2EE client container can explicitly demarcate transaction boundaries as described in ″Using component-managed transactions″ in the information center. 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″ in the information center.

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.

Extended JTA support: Extended JTA support provides application programming interfaces additional to the UserTransaction interface that is defined in the JTA as part of the J2EE specification. Specifically, the API extensions provide the following functionality:

- Access to global and local transaction identifiers associated with the thread.

  The global id is based on the tid in CosTransactions::PropagationContext: and the local id identifies the transaction uniquely within the local JVM.
- A transaction synchronization callback that enables any J2EE component to register an interest in transaction completion.

  This can be used by advanced applications to flush updates before transaction completion and clear up state after transaction completion. J2EE (and related) specifications position this function generally as the domain of the J2EE containers.

An application uses a JNDI lookup of java:comp/websphere/ExtendedJTATransaction to get an ExtendedJTATransaction object, which it then uses as follows:

```
ExtendedJTATransaction exJTA = (ExtendedJTATransaction)ctx.lookup("
 java:comp/websphere/ExtendedJTATransaction");
SynchronizationCallback sync = new SynchronizationCallback();
exJTA.registerSynchronizationCallback(sync);
```

The ExtendedJTATransaction object supports the registration of one or more application-provided SynchronizationCallbacks. Depending on how the callback is registered, each registered callback is called at one of the following points:

- At the end of every transaction that runs on the application server (whether the transaction is started locally or imported).

- At the end of the transaction for which the callback was registered.

The following information provides an overview of the interfaces provided by the Extended JTA support. For more detailed information, see the Javadoc.

SynchronizationCallback interface

An object implementing this interface is enlisted once through the ExtendedJTATransaction interface, and receives notification of transaction completion.

Although an object implementing this interface can run in a J2EE server, there is no specific J2EE component active when this object is called. So, the object has limited direct access to any J2EE resources. Specifically, it has no access to the java: namespace or to any container-mediated resource. Such an object can cache a reference to a J2EE component (for example, a stateless session bean) that it delegates to. The object would then have all the normal access to J2EE resources and could be used, for example, to acquire a JDBC connection and flush updates to a database during beforeCompletion.

ExtendedJTATransaction interface

A WebSphere programming model extension to the J2EE JTA support. An object implementing this interface is bound, by WebSphere J2EE containers that support this interface, at java:comp/websphere/ExtendedJTATransaction. Access to this object, when called from an EJB container, is not restricted to component-managed transactions.

Web Services – Atomic Transaction 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 9, shows a transaction context shared between two WebSphere application servers for a Web service request that contains a WS-AT CoordinationContext.

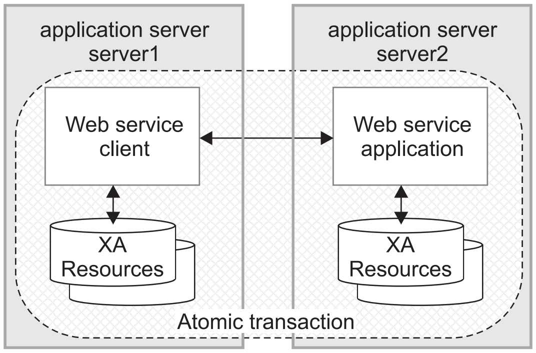


Figure 9. Transaction context shared between two WebSphere application servers.

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.

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″ in the information center.

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.

Extend the duration of a local transaction beyond the duration of an EJB component method.

The J2EE specifications restrict the use of RMLTs to single EJB methods. This restriction is because the specifications have no scoping device, beyond a container-imposed method boundary, to which an RMLT can be extended. You can exploit the **Local Transactions - Boundary** extended deployment setting to give the following advantages:

- Significantly extend the use-cases of RMLTs
- Make conversational interactions with one-phase resource managers possible through ActivitySession support.

You can use an ActivitySession to provide a distributed context with a boundary that is longer than a single method. You can extend the use of RMLTs over the longer ActivitySession boundary, which can be controlled by a client. The ActivitySession boundary reduces the need to use distributed transactions where ACID operations on multiple resources are not needed. This benefit is exploited through the **Local Transactions - Boundary** extended deployment setting of `ActivitySession`. Such extended RMLTs can remain under the control of the application or be managed by the container depending on the use of the **Local Transactions - Resolution-control** deployment descriptor setting.

Coordinate multiple one-phase resource managers.

For resource managers that do not support XA transaction coordination, a client can exploit ActivitySession-bounded local transaction contexts. Such contexts give a client the same ability to control the completion direction of the resource updates by the resource managers as the client has for transactional resource managers. A client can start an ActivitySession and call its entity beans under that context. Those beans can perform their RMLTs within the scope of that ActivitySession and return without completing the RMLTs. The client can later complete the ActivitySession in a commit or rollback direction and cause the container to drive the ActivitySession-bounded RMLTs in that coordinated direction.

To determine how best to configure the transaction support for an application, depending on what you want to do with transactions, consider the following points.

General points

- You want to start and end global transactions explicitly in the application (BMT session beans and servlets only).

  For a session bean, set the **Transaction type** to `Bean` (to use bean-managed transactions) in the component's deployment descriptor. (You do not need to do this for servlets.)
- You want to access only one XA or non-XA resource in a method.

  In the component's deployment descriptor, set **Local Transactions - Resolution-control** to `ContainerAtBoundary`. In the Container transaction deployment descriptor, set **Transaction** to `Supports`.
- You want to access several XA resources atomically across one or more bean methods.

  In the Container transaction deployment descriptor, set **Transaction** to `Required`, `Requires new`, or `Mandatory`.
- You want to access several non-XA resource in a method without having to worry about managing your own local transactions.

  In the component's deployment descriptor, set **Local Transactions - Resolution-control** to `ContainerAtBoundary`. In the Container transaction deployment descriptor, set **Transaction** to `Not supported`.
- You want to access several non-XA resources in a method and want to manage them independently.

  In the component's deployment descriptor, set **Local Transactions - Resolution-control** to `Application` and set **Local Transactions - Unresolved-action** to **Rollback**. In the Container transaction deployment descriptor, set **Transaction** to `Not supported`.
- You want to access one of more non-XA resources across multiple EJB method calls without having to worry about managing your own local transactions.

  In the component's deployment descriptor, set **Local Transactions - Resolution-control** to `ContainerAtBoundary`, **Local Transactions - Boundary** to `ActivitySession`, and **Bean Cache - Activate at** to `ActivitySession`. In the Container transaction deployment descriptor, set **Transaction** to `Not supported` and set **ActivitySession** attribute to `Required`, `Requires new`, or `Mandatory`.
- You want to access several non-XA resources across multiple EJB method calls and want to manage them independently.

  In the component's deployment descriptor, set **Local Transactions - Resolution-control** to `Application`, **Local Transactions - Boundary** to `ActivitySession`, and **Bean Cache - Activate**

at to ActivitySession. In the Container Transaction deployment descriptor, set **Transaction** to Not supported and set **ActivitySession** attribute to Required, Requires new, or Mandatory.

- You want to use a single non-XA resource and one or more XAResources.

  Use the Last Participant Support.

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.

Managing active and prepared transactions

Use this task to manage active and prepared transactions that might need administrator action.

Under normal circumstances, transactions should run and complete (commit or rollback) automatically, without the need for intervention. However, in some circumstances, you may need to resolve a transaction manually. For example, you may want to rollback a transaction that has become stuck polling a resource manager that you know will not become available again within the desired timeframe.

Note: If you choose to complete a transaction on an application server, it is recorded as having completed in the transaction service logs for that server, so will not be eligible for recovery during server start up. If you complete a transaction, you are responsible for cleaning up any in-doubt transactions on the resource managers affected.

You can use the administrative console to display a snapshot of all the transactions in an application server that are in the following states:

Manual transactions
Transactions awaiting administrative completion. For each transaction, the local or globalid 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 or globalid 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 or globalid 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 or globalid is displayed. You can choose to display information on each resource (specifically, which resource manager it is associated with) associated with the transaction. You can also choose to commit or rollback transactions in this state.

To manage the active and prepared transactions for an application server, use the administrative console to complete the following steps:

1. Display the Transaction Service runtime page for application server:
   a. In the navigation pane, click **Servers-> Application Servers**
   b. In the content pane, click the name of the application server
   c. In the content pane, click the **Runtime** tab.
   d. Under Additional Properties, click **Transaction Service**

   This displays values for the runtime properties of the transaction service, including the number of transactions in the active and prepared states.

2. To display a snapshot of the transactions in a specific state, click **Review** in the field label.
3. **Optional:** If you want to display information about the resources associated with a transaction, click the name of the transaction.
4. **Optional:** If you want to act on a transaction, select the check box provided on the entry for the transaction, then click one of the buttons provided. Alternatively, to act on all transactions, select the check box in the header of the transactions table, then click a button.

Managing transaction logging for optimum server availability

This topic describes some considerations and actions that you can use to manage transaction logging to help ensure that the availability of your application servers is optimized.

The transaction service writes information to the transaction log for every global transaction which involves two or more resources or is distributed across multiple servers. The transaction log is stored on disk and is used by the transaction service for recovery after a system or server crash. The transaction log for each application server consists of multiple subdirectories and files held in a single directory. You can change the directory that an application server uses to store the transaction log, as described in Configuring transaction properties for an application server.

When a global transaction is completed, the information in the transaction log is not needed anymore so is marked for deletion. Periodically, this redundant information is garbage collected and the space reused by new transactions. The log files are created of fixed size at server startup, thus no further disk space allocation is required during the lifetime of the server. The default allocation is suitable for around 4000 concurrent transactions.

If all the log space is in use when a transaction needs to save information, the transaction is rolled back and the message `WTRN0083W: The transaction log is full. Transaction rolled back.` is reported to the system error log. No more transactions can commit until more log space is made available when existing active transactions complete.

You can monitor the number of concurrent global transactions by using the performance monitoring counters for transactions. The “Global transaction commit time” counter is a measure of how long a transaction takes to complete and, therefore, how long the log is in use by a transaction. If this value is high, then transactions are taking a long time to complete, which can be due to resource manager or network failures. If you ensure this value is low, the log is more efficiently used and unlikely to become full.

You can change the default size of log files by updating the transaction log settings as described in Configuring transaction properties for an application server.

Configuring transaction aspects of servers for optimum availability

This topic describes some considerations and actions that you can take to configure transaction-related aspects of application servers, to optimize the availability of those servers.

This helps your transactions to complete or recover more quickly. After changing transaction-related properties of an application server, you must restart the server.

To configure transaction-related aspects of application servers for optimum availability, complete the following steps:

1. Store the transaction log files on a fast disk in a highly-available file system, such as a RAID device. The transaction log may need to be accessed by every global transaction and be used for transaction recovery after a crash. Therefore, the disk the log files are being written to should be on a highly-available file system, such as a RAID device.

   The performance of the disk also directly affects the transaction performance. In general, a global transaction makes two disk writes, one after the prepare phase when the outcome of the transaction is known (this information is forced to disk) and a further disk write at transaction completion. Therefore, the transaction logs should be placed on the fastest disks available and not make use of network mounted devices.

2. Mirror the transaction log files by using hardware disk mirroring or dual-ported disks. If log files have been mirrored or can be recovered, they can be used when restarting a failed server or moved to an another machine and another server started there to perform recovery.

   Hardware disk mirroring or dual-ported disks can be used by specifiying the appropriate file system directory for the transaction logs using the WebSphere Administrative Console.

3. Specify the optimum location of the transaction log directory for application servers. By default, an application server places transaction log files in a subdirectory of the installed WebSphere Application Server, where the subdirectory name is the same as the server name. For example, the default directory for an application server named `server1` on Windows is `c:\WebSphere\AppServer\profiles\`*`profilename`*`\tranlog\server1`.

   You can define a specific location for the transaction log directory for an application server by setting the **Transaction Log Directory** property for the server. If the directory for the transaction logs has not been created at application server start up, the directory structure is created for you.

   Note: If you change the transaction log directory, you should apply the change and restart the application server as soon as possible, to minimize the risk of problems caused that might occur before the application server is restarted. For example, if a problem causes the server to fail (with in-flight transactions), the server next starts with the new log directory and is unable to automatically resolve in-flight transactions that were recorded in the old log directory.

4. Never allow more than one application server to concurrently use the same set of log files. Because the transaction logs record the state of global transactions within a server, if the logs become lost or corrupt, then transactions that are in the prepared state before failure can leave resources in an in-doubt state and prevent further updates or access to the resources by other users or servers. These transactions may need to be manually resolved by either committing or rolling back the transactions at the affected resource managers. The failed server can then be cold-started, which creates new empty transaction logs.

   If log files have been mirrored or can be recovered, they can be used when restarting the failed server or moved to an alternate server or machine and another server restarted to perform recovery, as described in the related tasks.

   Never allow more than one application server to concurrently use the same set of log files, because each server will destroy the information recorded by the other, resulting in corrupt log files that are unusable for future recovery purposes.

5. Configure application servers to always use the same listening port address at each startup. If you are running distributed transactions between multiple application servers, the remote object references saved in the transaction log need to be redirected to the originating server on recovery.

   On WebSphere Application Server Network Deployment, the node agents automatically redirect such remote object references to the appropriate application servers on recovery. However, if the distributed transaction is between application servers that are not on WebSphere Application Server Network Deployment, then you must handle the redirection of remote object references for transaction recovery to complete. For example, you must do this is if an application server is deployed on WebSphere Application Server (not the Network Deployment edition) and runs distributed transactions with non-WebSphere EJB or Corba servers.

   In particular, the default restart action of an application server not on Application Server Network Deployment is to use a different listening port address to the port when the server shut down. This prevents transaction recovery completing. To overcome this, you should always configure application servers to always use the same listening port address at each startup (see the ORB property com.ibm.CORBA.ListenerPort in ORB service settings that can be added to the administrative console). You may need to make similar configuration changes to other application servers involved in transactions, to be able to access those servers during recovery.

Moving a transaction log from one server to another

This topic describes some considerations and actions that you can take to move the transaction logs for an application server to another server.

To move transaction logs from one application server to another, consider the following steps:

1. Move all the transaction log files for the application server. The transaction log directory for each server contains a number of files and subdirectories. When moving transaction logs from one server to another you must move all of the files and subdirectories together as a single unit; otherwise recovery may not complete resulting in data inconsistency.
2. For a server configuration where there are no distributed transactions, move the transaction logs to any server that has access to the same resource managers. For a single server or network-deployed server configuration where it is known there are no distributed transactions present in the logs, the transaction logs can be moved to any server (on any node) that has access to the same resource managers as the original server. For example, the server needs communication and valid security access to databases or message queues.

   All the transaction log files for the original server need to be moved to a directory accessible by the new server. This can be accomplished by either renaming the transaction log directory or copying all thecontents to the new server's transaction log directory before starting the new server.

   Note: To complete transaction recovery, the application server uses the resource manager configuration information in the transaction logs. However, for the application server to continue to do new work with the same resource managers, the server must have an appropriate resource manager configuration (as for the original server).
3. For a network-deployed server configuration where there are distributed transactions, move the transaction logs to a server that has the same name and host IP address, and access to the same resource managers. For a network-deployed server configuration, when it is known there are distributed transactions present in the logs, there are more restrictions. Distributed transactions that access multiple servers log information about each server involved in the transaction. This information includes the server name and the IP address of the machine on which the server is running. When recovery is taking place on server restart, the server uses this information to contact the distributed servers and similarly, the distributed servers try to contact the server with the same original name. So, if a server fails and the logs need to the recovered on an alternative server, that alternative server needs to have the same name and host IP address as the original server. The alternative server also needs to have access to the same resource managers as the original server. For example, the server needs communication and valid security access to databases or message queues.

   Note: All servers within a cell must have unique names.

   Note: To complete transaction recovery, the application server uses the resource manager configuration information in the transaction logs. However, for the application server to continue to do new work with the same resource managers, the server must have an appropriate resource manager configuration (as for the original server).

Restarting an application server on a different host

This topic describes some considerations and actions that you can take with transaction logs to restart an application server on a different host.

Moving transactions logs to a different host is similar to moving logs from one server to another, as described in Moving transaction logs from one server to another.

This involves moving an original application server on one host to an alternative server, which has access to the same resource managers, on another host. For a network-deployed server configuration, the alternative server must have the same name and host IP address as the original server.

Note: To complete transaction recovery, the application server uses the resource manager configuration information in the transaction logs. However, for the application server to continue to do new work with the same resource managers, the server must have an appropriate resource manager configuration (as for the original server).

To restart an application server on a different host, complete the following steps:

1. Ensure that the alternative application server is stopped.
2. Move all the transaction logs for the original server to the alternative application server, according to the considerations described in Moving transaction logs from one server to another.
3. Restart the alternative application server.

Interoperating transactionally between application servers

This topic describes some considerations and actions that you can take to interoperate transactionally between different types of application servers.

WebSphere Application Server is a transaction manager that supports transactional interoperation with other transaction managers through either the CORBA Object Transaction Service (OTS) protocol (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.

To interoperate transactionally, using the OTS protocol, between WebSphere Application Server version 6 and versions of the WebSphere Application Server before version 5.0, you need to set the following system properties on application servers that are before version 5.0:

```
com.ibm.ejs.jts.jts.ControlSet.nativeOnly=false
com.ibm.ejs.jts.jts.ControlSet.interoperabilityOnly=true
```

For example, if you want to interoperate between application servers at WebSphere Application Server version 6 and WebSphere Application Server version 4.0.n, you need to set the system properties on the version 4.0.n application servers.

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. See ″Assembly tools″ in the information center.

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″ in the information center.

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. See ″Starting an assembly tool″ in the information center.
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** check box.
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. See ″Verifying archive files″ in the information center
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: 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.

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**

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. 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.
3. Select the Configuration tab, to display the transaction-related configuration properties.
4. Select the **Enable logging for heuristic reporting** checkbox.
5. Click **OK**.
6. 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

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″ in the information center 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.

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	Selected
Range	**Selected** When the application server starts, it attempts to start the ActivitySession service automatically. **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 start the service manually or select this property then restart the server.

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
  - ActivitySessions and transaction contexts
  - Using ActivitySessions with HTTP sessions
- The ActivitySession service programming interfaces
- Developing a J2EE application to use ActivitySessions
- ActivitySessions samples
- Setting Web module ActivitySession deployment attributes
- Setting Web module ActivitySession deployment attributes
- Disabling or enabling the ActivitySession service
- Configuring the default ActivitySession timeout
- Troubleshooting ActivitySessions

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:

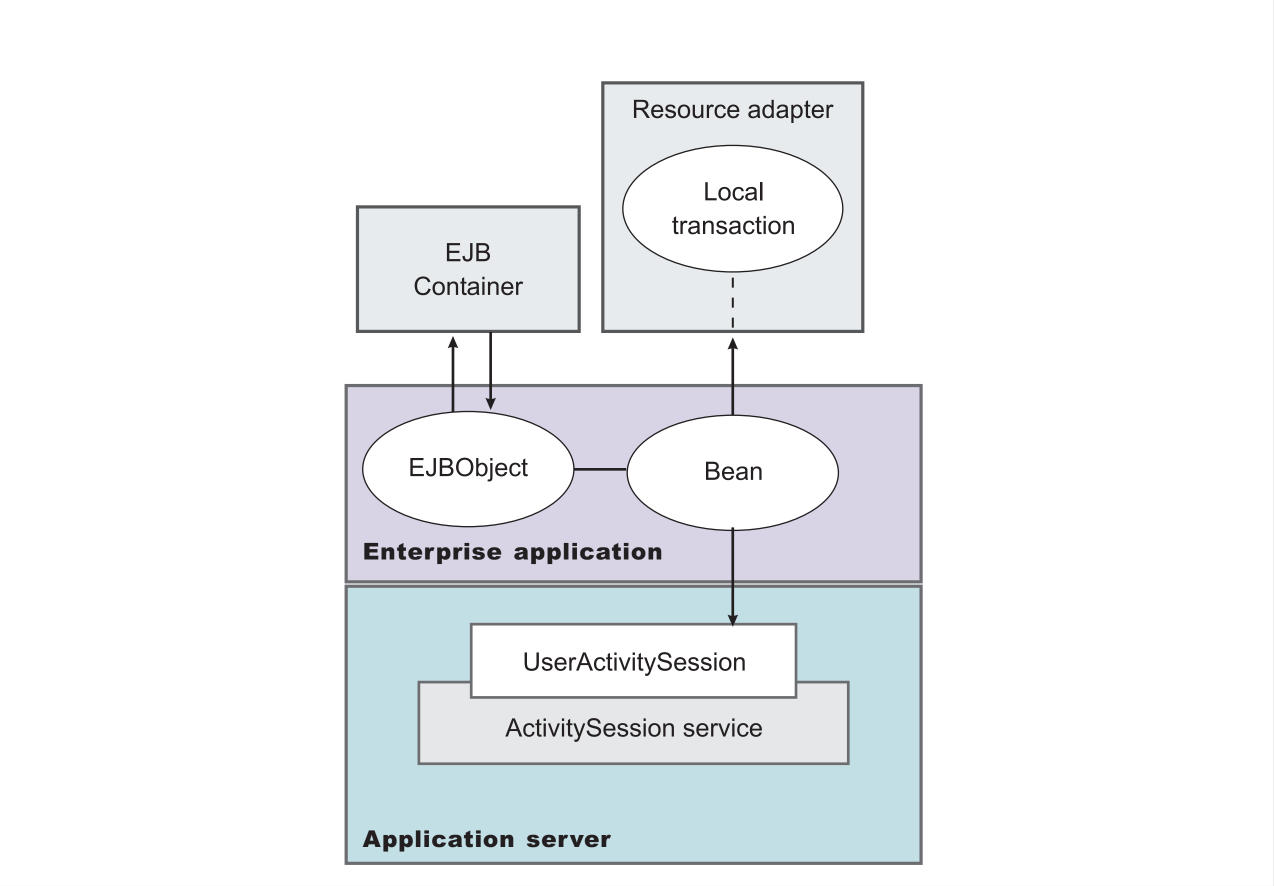


Figure 10. The ActivitySession service. This figure show the main components of the ActivitySession service within WebSphere Application server. For an overview of these components, see the text that accompanies this figure.

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.

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.

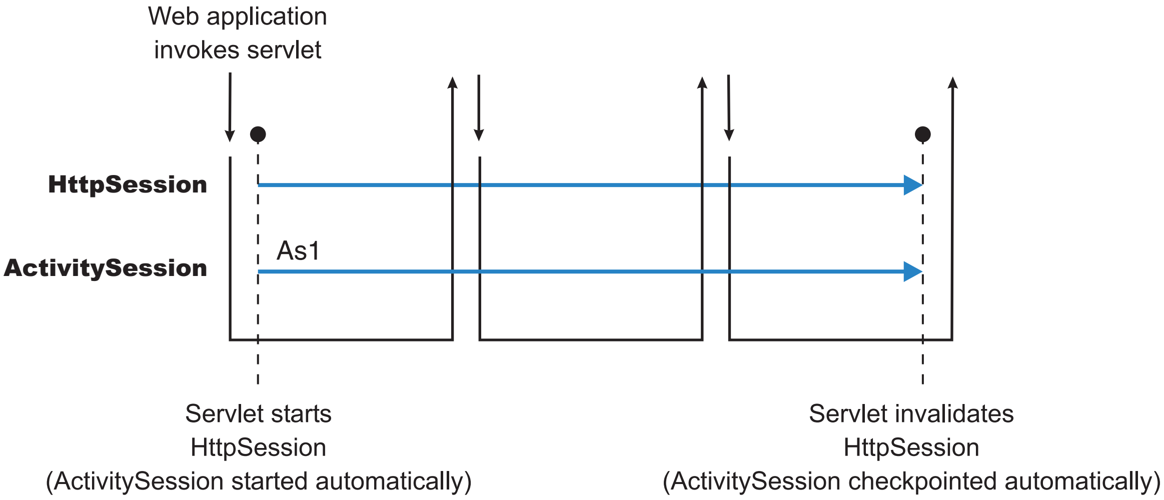
An ActivitySession is associated automatically with an HttpSession, so can be used to extend access to the ActivitySession over multiple HTTP invocations, over inclusion or forwarding of servlets, and to support EJB activation periods that can be determined by the lifecycle of the web HTTP client. An 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:


Web application invokes servlet
HttpSession
ActivitySession
As1
Servlet starts HttpSession (ActivitySession started automatically)
Servlet invalidates HttpSession (ActivitySession checkpointed automatically)

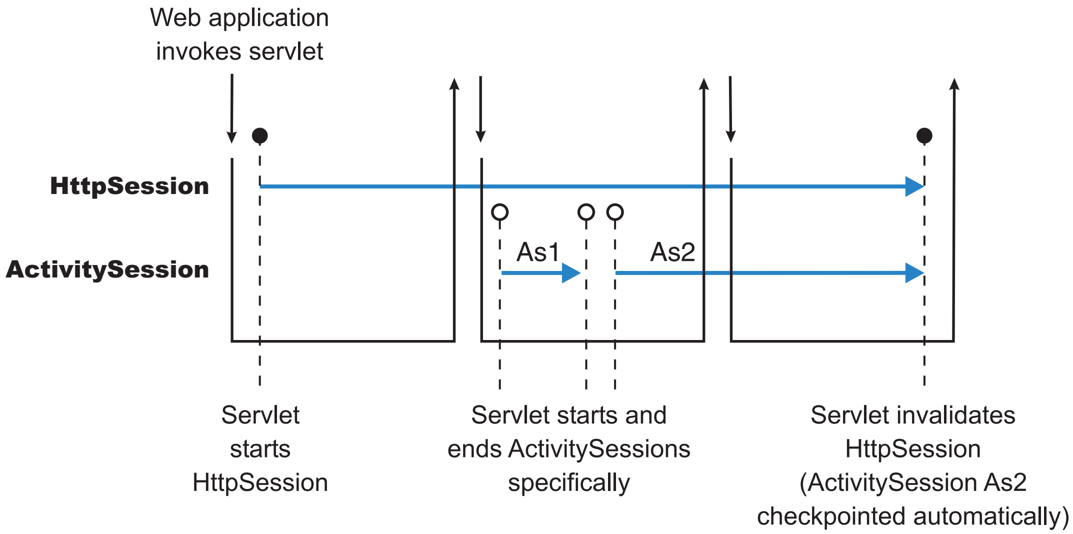
- The Web application starts and ends ActivitySessions.

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:


Web application invokes servlet
HttpSession
ActivitySession
As1
As2
Servlet starts HttpSession
Servlet starts and ends ActivitySessions specifically
Servlet invalidates HttpSession (ActivitySession As2 checkpointed automatically)

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[3]. 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.

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 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

[3] 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

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.

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 11. 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		

Table 11. 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
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		
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	

Table 11. 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
S1	Suspend S1, Start S2		
T1	Suspend T1, Start S1		
S1, T1	Suspend S1 + T1, Start S2		
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		
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	

Table 11. 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
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		
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	

Table 11. 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
S1	No Action		
T1	Exception		
S1,T1	Exception		
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		

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”
- “Programming specifications” on page 1872
- “Other” on page 1872

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

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

   [Default] 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

   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.

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. Manage application profiles. This topic describes how to add and remove tasks from application profiles using the administrative console.
3. Use 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.

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.

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 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.

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.

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″ in the information center.

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.

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.

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 in the information center:
   - 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.

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**.

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**.

Name:

The name of the application profile.

The name must be unique; multiple profiles cannot share the same name.

Data type	String

Description:

A description of the application profile.

Data type	String

Application profile settings:

Use this page to modify application profile settings.

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

Name:

The name of the application profile.

The name must be unique; multiple profiles cannot share the same name.

Data type	String

Description:

A description of the application profile.

Data type	String

Task collection:

Use this page to manage tasks.

Requests associated with any of the configured tasks operate under the access-intent policies that are configured with the profile. A task can be configured on only *one* application profile.

To view this administrative console page, click **Applications** > **Enterprise Applications** > *application_name* > **Application Profiles** > *application_profile_name* > **Tasks**.

Name:

The name of the task.

The task name must be unique among the set of application profiles.

Data type	String

Description:

A description of the task.

Data type	String

Task settings:

Use this page to modify task settings.

To view this administrative console page, click **Applications** > **Enterprise Applications** > *application_name* > **Application Profiles** > *application_profile_name* > **Tasks** > *task_name*.

Name:

The name of the task.

The task name must be unique among the set of application profiles.

Data type	String

Description:

A description of the task.

Data type	String

Asynchronous beans

Using asynchronous beans

The asynchronous beans feature adds a new set of APIs that enable Java 2 Platform Enterprise Edition J2EE applications to run asynchronously inside an Integration Server. This topic provides a brief overview of the tasks involved in using asynchronous beans. For a more detailed description of the asynchronous beans model, review the conceptual topic Asynchronous beans. For detailed information on the programming model for supported asynchronous beans interfaces, see the topic Work managers.

1. Configure work managers.
2. Configure timer managers.
3. Assemble applications that use asynchronous beans work managers.
4. Develop work objects to run code in parallel.
5. Develop event listeners.
6. Develop asynchronous scopes.

Asynchronous beans:

An asynchronous bean is a Java object or enterprise bean that can 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″ in the information center). 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″ in the information center.

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″ in the information center. 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. Any work manager that implements the `java.io.Serializable` does not have the remote start capability.

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 depends 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 12. Look up work manager

Asynchronous beans	CommonJ

`InitialContext ctx  = new InitialContext();`
`com.ibm.websphere.asynchbeans.WorkManager wm  =`
`(com.ibm.websphere.asynchbeans.WorkManager)`
`        ctx.lookup("java:comp/env/wm/MyWorkMgr");`
	`InitialContext ctx  = new InitialContext();`
	`commonj.work.WorkManager wm =`
	`  (commonj.work.WorkManager)`
	`     ctx.lookup("java:comp/env/wm/MyWorkMgr");`

Table 13. 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 14. Submit the work

Asynchronous beans	CommonJ

Table 14. Submit the work (continued)

MyWork work1 =
 new MyWork(new URI("http://www.example./com/1");
MyWork work2 =
 new MyWork(new URI("http://www.example./com/2");
 	
WorkItem item1;
WorkItem item2;
Item1=wm.startWork(work1);
Item2=wm.startWork(work2);
 	
// case 1: block until all items are done
ArrayList col1 = new ArrayList();
Col1.add(item1);
Col1.add(item2);
wm.join(col1, WorkManager.JOIN_AND,
 (long)WorkManager.IMMEDIATE);
// when the works are done
System.out.println("work1 data="+work1.getData());
System.out.println("work2 data="+work2.getData());
 	
// case 2: wait for any of the items to complete.
Boolean ret = wm.join(col1,
  WorkManager.JOIN_OR, 1000);
	MyWork work1 =
	 new MyWork(new URI("http://www.example./com/1");
	MyWork work2 =
	 new MyWork(new URI("http://www.example./com/2");
	
	WorkItem item1;
	WorkItem item2;
	Item1=wm.schedule(work1);
	Item2=wm.schedule(work2);
	
	// case 1: block until all items are done
	Collection col1 = new ArrayList();
	col1.add(item1);
	col1.add(item2);
	wm.waitForAll(col1, WorkManager.IMMEDIATE);
	// when the works are done
	System.out.println("work1 data="+work1.getData());
	System.out.println("work2 data="+work2.getData());
	
	// case 2: wait for any of the items to complete.
	Collection finished = wm.waitForAny(col1,1000);
	// 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());
	  }
	 }
	}

Timer manager examples

Table 15. Create a timer manager

Asynchronous beans	CommonJ

InitialContext ctx = new InitialContext();
com.ibm.websphere.asynchbeans.WorkManager wm =
 (com.ibm.websphere.asynchbeans.WorkManager)
   ctx.lookup("java:comp/env/wm/MyWorkMgr");
 	
AsynchScope ascope;
 try {
   Ascope = wm.createAsynchScope("ABScope");
 } Catch (DuplicateKeyException ex)
 {
    Ascope = wm.findAsynchScope("ABScope");
    ex.printStackTrace();
 }
 	
 // get an AlarmManager
 AlarmManager aMgr= ascope.getAlarmManager();
	InitialContext ctx = new InitialContext();
	Commonj.timers.TimerManager tm =
	  (commonj.timers.TimerManager)
	    ctx.lookup("java:comp/env/tm/MyTimerManager");

Table 16. Fire the timer

Asynchronous beans	CommonJ

Table 16. Fire the timer (continued)

// create alarm
ABAlarmListener listener = new ABAlarmListener();
Alarm am =
 aMgr.create(listener, "SomeContext", 1000*60);
	// create Timer
	TimerListener listener =
	 new StockQuoteTimerListener(
	  "qqq", "johndoe@example.com");
	Timer timer = tm.schedule(listener, 1000*60);
	
	// Fixed-delay: schedule timer to expire in 60 seconds
	// from now and repeat every hour thereafter.
	Timer timer = tm.schedule(listener, 1000*60, 1000*30);
	
	// Fixed-rate: schedule timer to expire in 60 seconds
	// from now and repeat every hour thereafter
	Timer timer =
	 tm.scheduleAtFixedRate(listener, 1000*60, 1000*30);

Timer managers: The timer manager combines the functions of the asynchronous beans alarm manager and asynchronous scope. So, when a timer manager is created, it internally uses an asynchronous scope to provide the timer manager life cycle functions. You can look up the timer manager in the JNDI name space. This capability is different from the alarm manager that is retrieved through the asynchronous beans scope. Each lookup of the timer manager returns a new logical timer manager that can be destroyed independently of all other timer managers.

A timer manager can be configured with a number of thread pools through the administrative console. For deployment you can bind this timer manager to a resource reference at assembly time, so the resource reference can be used by the application to look up the timer manager.

The Java code to look up the timer manager is:

```
InitialContext ic = new InitialContext();
TimerManager tm = (TimerManager)ic.lookup("java:comp/env/tm/TimerManager");
```

The programming model for setting up the alarm listener and the timer listener is different. The following code example shows that difference.

Table 17. Set up the timer listener

Asynchronous beans	CommonJ

public class ABAlarmListener
   implements AlarmListener {
 public void fired(Alarm alarm) {
  System.out.println(
   "Alarm fired. Context ="
    +alarm.getContext());
 }
}
	public class StockQuoteTimerListener implements TimerListener {
	 String context;
	 String url;
	 public StockQuoteTimerListener(String context, String url){
	  this.context = context;
	 this.url = url;
	 }
	 public void timerExpired(Timer timer) {
	  System.out.println("Timer fired. Context ="+
	((StockQuoteTimerListener)timer.getTimerListener()).getContext());
	 }
	 public String getContext() {
	  return context;
	 }
	}

Example: Asynchronous bean connection management:

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 in the information center.) 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...
 }
}
```

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**.

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

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.

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**.

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.

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 distributed platforms, WebSphere Application Server, Version 4.0, supported the configuration of dynamic servlet caching through the `servletcache.xml` file. To utilize the new and improved functionality of the dynamic cache service, configure your cache policy using the `cachespec.xml` format. See ″Configuring cacheable objects with the cachespec.xml file″ 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″ in the information center.
2. Configure the type of caching that you are using.
   - Configure servlet caching.
   - Configure Edge Side Include caching.
   - Configure command caching.
   - “Example: Caching Web services” on page 1902.
   - Configure the Web services client cache.
3. You can monitor the results of your configuration using the dynamic cache monitor. For more information, see ″Displaying cache information″ in the information center.
4. If you have any problems with your configuration, see ″Troubleshooting the dynamic cache service″ in the information center.

To use the DistributedMap and DistributedObjectCache interfaces for the dynamic cache, see “Using the DistributedMap and DistributedObjectCache interfaces for the dynamic cache” on page 1936.

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 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>
```

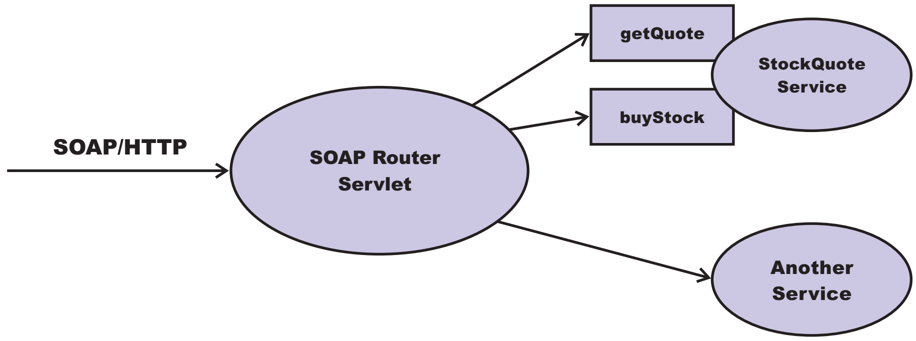
The SOAPAction HTTP header in the request is defined in the SOAP specification and is used by HTTP proxy servers to dispatch requests to particular HTTP servers. WebSphere Application Server dynamic cache can use this header in its cache policies to build IDs without having to parse the SOAP message.

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 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>
</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>
</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″ 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 1908 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 1909 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 1902
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.

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 setting for an environment without clustering is **Not Shared**. When enabling replication, the default value is **Push only**.

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 to the dynamic cache.

To enable servlet caching, you must complete “Enabling the dynamic cache service” on page 1906.
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” on page 113 for more information.

Define the cache policy for your servlets by configuring cacheable objects with the cachespec.xml file.

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 the URI 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″ in the information center. To configure cache policies for your servlets, see ″Configuring cacheable objects with the cachespec.xml file.″

Configuring dynamic cache disk offload:

Use this task to configure dynamic cache disk offload, which saves cache entries that are deleted from the memory cache to disk.

By default, when the number of cache entries reaches the configured limit for a given application server, cache entries are removed from the memory cache, allowing newer entries to be stored in the cache. Use disk offload to copy the cache entries that are being removed from the memory cache to disk for potential future access.

1. In the administrative console, click **Servers** > **Application servers >***server_name* **> Container services > Dynamic cache service**.
2. Select **Enable disk offload**. 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.
3. Click **Apply** or **OK**.
4. Restart WebSphere Application Server.

Application servers must have different disk offload locations

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 nonroot, 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 nonroot servers to crash when they try to read or write to the cache.

On distributed platforms, the disk offload location must be unique for servers 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. Change the disk offload location by clicking **Servers** > **Application servers >***server_name* **> Container services > Dynamic cache service** in the administrative console.

Java virtual machine custom properties for tuning the disk cache:

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. If you want to set the custom properties on the default cache instance, however, you must use 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

Specifies 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

Specifies if extra memory buffers are 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 it 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 distributed platforms, the ESI processor's cache can be monitored through the CacheMonitor application. In order for ESI processor's cache to be visible in the CacheMonitor, the DynaCacheEsi application must be installed as described above and the ESIInvalidationMonitor property must be set to true in the `plugin-cfg.xml` file.

When WebSphere Application Server is used to serve static data, such as images and HTML on the application server, the URLs are also cached in the ESI processor. This data has a default timeout of 300 seconds. You can change the timeout value by adding the property com.ibm.servlet.file.esi.timeOut to the Java virtual machine (JVM) command line parameters. The following example shows how to set a one minute timeout on static data cached in the plug-in:

```
-Dcom.ibm.servlet.file.esi.timeOut=60
```

For information about configuring alternate URL, see ″"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″ in the information center.

Configuring external cache groups:

The dynamic cache can control caches outside of the application server, such as the Edge server, an IBM HTTP Server for distributed platforms, or an HTTP Server ESI Fragment Processor plug-in for distributed platforms. 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 URIs and pushes matching pages to the external cache. The entries can then be served from the external cache, instead of 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
WTE
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.

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
WTE
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. On distributed platforms, turn on servlet caching for each application server that uses the cache accelerator.
   b. Configure an external cache group on the application server:
      1) Click **Servers > Application servers >** *server_name* **> Container services > Dynamic cache service > External cache groups**.
      2) Click **New** on the External cache group administrative console page to define an external cache group named `afpa` for each application server that uses the cache accelerator.
      3) In the **External cache group** field, type `afpa` and apply the changes.
   c. Add a member to the group with an adapter bean name of `com.ibm.ws.cache.servlet.Afpa`.
      1) Click **Afpa > External cache group members**.
      2) Click **New** on the External cache group members administrative console page.
      3) In the **AdapterBean name** field, type `com.ibm.ws.cache.servlet.Afpa`.
      4) In the **Address** field, enter an unused port number.
   d. Add a cache policy in the `cachespec.xml` file for the servlet or JSP file you want to cache. Add the following property to the cache policy:

      ```
      <property name="ExternalCache">afpa</property>
      ```
2. Enable cache accelerator on the IBM HTTP Server for distributed platforms:
   a. Add the following directives to the end of the `httpd.conf` file:
      - `AfpaEnable`
      - `AfpaCache on`
      - `AfpaLogFile "`*`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 cacheable objects with the cachespec.xml file

Enable the dynamic cache. See ″Enabling the dynamic cache service″ in the information center.

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 1919 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).
   - 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 1919 for a list of *<component>* elements.
5. Invalidate other cache entries as a side effect of this object start, if relevant. You can define invalidation rules in exactly the same manner as dependency IDs. However, the IDs that are generated by invalidation rules are used to invalidate cache entries that have those same dependency IDs.

   The invalidation ID is generated by concatenating the invalidation ID base string with the values returned by its component element. If a required component returns a null value, then the entire invalidation ID is not generated and no invalidation occurs. Multiple invalidation rules can exist per cache-entry. All invalidation rules run separately.
6. Ensure your cache policy is working correctly. You can modify the policies within the `cachespec.xml` file while your application is running. The dynamic cache reloads the updated file automatically. If you are caching static content and you are adding the cache policy to an application for the first time, you must restart the application. You do not need to restart the application server to activate the new cache policy. See "Verifying the cacheable page" for more information.

Typically you declare several *<cache-entry>* elements inside a `cachespec.xml` file.

When new versions of the `cachespec.xml` are detected, the old policies are replaced. Objects that cached through the old policy file are not automatically invalidated from the cache; they are either reused with the new policy or eliminated from the cache through its replacement algorithm.

For each of the three IDs (cache, dependency, invalidation) generated by cache entries, a *<cache-entry>* can contain multiple elements. The dynamic cache runs the *<cache-id>* rules in order, and the first one that successfully generates an ID is used to cache that output. If the object is to be cached, each one of the *<dependency-id>* elements is run to build a set of dependency IDs for that cache entry. Finally, each of the *<invalidation>* elements are run, building a list of IDs that the dynamic cache invalidates, whether or not this object is cached.

Verifying the cacheable page:

Use this task to verify that the dynamic cache service has its cache policies configured correctly and is serving cached content.

The dynamic cache service should be enabled. You should have a cache policy developed for your application. See ″Configuring cacheable objects with the cachespec.xml file″ in the information center. You must have servlet caching enabled in the web container. See ″Configuring servlet caching″ in the information center.

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
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″ in the information center.

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 thecachespec.xml file with the deployment module. Use an assembly tool to define the cacheable objects. See ″Assembling applications″ in the information center for more information. 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

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 serve 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>
```

where *name* is the fully qualified class name of the command, servlet, or Web service.

There are two ways to use <name> 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 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 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 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. In single server environments, not-shared is the only valid value. When enabling replication, the default value is `shared-push` only. This property does not affect distribution to Edge Side Include processors through the Edge fragment caching property.

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 executes 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.
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

Type	Valid classes	Meaning
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:name OfOperation
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 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 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 1901. 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″ 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 1936.

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 the information center. 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″ in the information center.

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″ in the information center.

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″ in the information center for more information.

The DistributedMap and DistributedObjectCache interfaces are a 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 1939.There are three methods for configuring and using cache instances.

Note: For more information about the DistributedMap and DistributedObjectCache interfaces, see the API documentation for the com.ibm.websphere.cache package. See Javadoc for more information.

- **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.

Note: DistributedMap and DistributedObjectCache do not have authorization or access control associated with the cache entries.

Object cache instance settings:

An object cache instance is a location, in addition to the default shared dynamic cache, where any Java 2 Platform, Enterprise Edition (J2EE) application can store, distribute, and share data. This gives applications greater flexibility and better tuning of the cache resources. Use the DistributedMap programming interface to access this cache instance. See the API documentation for more information.

To view this administrative console page, click **Resources > Cache instances > Object cache instances >** *cache_instance_name*.

Name:

Specifies the required display name for the resource.

JNDI name:

Specifies the Java Naming and Directory Interface (JNDI) name for the resource. Use this name when looking up a reference to this cache instance. The results return a `DistributedMap` object.

Description:

Specifies a description for the resource. This field is optional.

Category:

Specifies a category string to classify or group the resource. This field is optional.

Cache size:

Specifies a positive integer for the maximum number of entries the cache holds. The cache size is usually in the thousands.

Default	2000
Range	100 - 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 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

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

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**.

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 not be available for cache instances that were created with a previous version of WebSphere Application Server.

Replication settings:

Click **Replication settings** to go to the replication settings panel to configure data replication for this cache instance.

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 Javadoc 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.

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 709. Install the cache monitor onto the application server you are trying to monitor. Installing the cache monitor on the *admin_host* (port 906*x*) is more secure than installing it on the *default_host* (908*x*), and so it is preferable to install it onto the *admin_host*.
2. Configure the web container transport chain and host alias for the server with cache monitor installed.
   a. If you installed the cache monitor on the *admin_host* (port 906*x*), check if a web container transport chain has been created. Click **Application servers >***server_name* **> Web Container settings > Web container transport chains**. If a web container transport chain (port 906*x*) does not exist you must create a web container transport chain in the *admin_host* for this server. If you are using *server1*, a web container transport chain is installed by default for admin port 9060.
   b. Add a host alias for the port your server is using. Click **Environment > Virtual hosts >***host_type* **> Host aliases** and create a new **Host name** and **Port** to add to the list.
   c. You can then access the cache monitor using http://*your_host_name*:*your_port_number*/cachemonitor.

   Tip: You can find the port number in the SystemOut.log file. Look for message TCPC0001I or SRVE0171I.
3. Access the cache monitor using a Web browser and the URL http://*your host_name*:*your port_number*/cachemonitor, where *your port_number* is the port associated with the host on which you installed the cache monitor application.
4. Verify the list of cache instances that are shown. For each cache instance, you can perform the following actions:

   Tip: You must select the servlet cache instance that you want to monitor. If you do not use servlet cache instances by using <cache-instance> tags in your cachespec.xml file, all the content is in the **baseCache** instance.

   - View the Statistics page and verify the cache configuration and cache data. Click **Reset Statistics** to reset the counters.
   - View the Cache Policies page to see which cache policies are currently loaded in the dynamic cache. Click on a template to view the cache ID rules for the template.

- View the Cache Contents page to examine the contents that are currently cached in memory.
- View the Edge Statistics page to view data about the current ESI processors configured for caching. Click **Refresh Statistics** to see the latest statistics or content from the ESI processors. Click **Reset Statistics** to reset the counters.
- View the Disk Offload page to view content that is currently off-loaded from memory to disk.

When you are viewing contents on memory or disk, click on a template to view all entries for that template, click on a dependency ID to view all entries for the ID, or click on the cache ID to view all the data that is cached for that entry.

5. Use the cache monitor to perform basic operations on data in a cache instance.

Remove an entry from cache
Click **Invalidate** when viewing a cache entry.

Remove all entries for a certain dependency ID
Click **Invalidate** when viewing entries for a dependency ID.

Remove all entries for a certain template
Click **Invalidate** when viewing entries for a template.

Move an entry to the front of the Least Recently Used queue to avoid eviction
Click **Refresh** when viewing a cache entry.

Move an entry from disk to cache
Click **Send to Memory** when viewing a cache entry on disk.

Clear the entire contents of the cache
Click **Clear Cache** while viewing statistics or contents.

Clear the contents on the ESI processors
Click **Clear Cache** while viewing ESI statistics or contents.

Clear the contents of the disk cache
Click **Clear Disk** while viewing disk contents.

Cache monitor:

Cache monitor is an installable Web application that provides a real-time view of the current state of dynamic cache. You use it to help verify that dynamic cache is operating as expected. The only way to manipulate the data in the cache is by using the cache monitor. It provides a GUI interface to manually change data.

Cache monitor provides a way to:

- **Verify the configuration of dynamic cache**

  After you create multiple servlet cache instances in the administrative console, you can configure properties, including the maximum size of the cache and disk offload location on each cache instance, as well as advanced features such as controlling external caches. You can verify the configuration of the dynamic cache by viewing of the configured features and properties in the cache monitor.
- **Verify the cache policies**

  To cache an object, unique IDs must be generated for different invocations of that object. To create unique IDs for each object, provide rules for each cacheable object in the `cachespec.xml` file, found inside the Web module `WEB-INF` or `enterprise bean META-INF` directory. See "Cachespec.xml file" in the information center. Each cacheable object can have multiple cache ID rules that run in sequence until either a rule returns a cache ID or no more rules remain. If none of the cache ID generation rules produce a valid cache ID, then the object is not cached. There can be multiple `cachespec.xml` files with multiple cache ID rules. With cache monitor, you can verify the policies of each object. You can also view all of the cache polices for each cache instance that is currently loaded in dynamic cache. This view is also convenient to verify that the `cachespec.xml` file was read by the dynamic cache without errors.
- **Monitor cache statistics**

  You can view the essential cache data, such as number of cache hits, cache misses, and number of entries in each cache instance. With this data, you can tune the cache configuration to improve the dynamic cache performance. For example, if the number of used entries is often high, and entries are being removed and recreated, consider increasing the maximum size of the cache or enabling disk offload.

- **Monitor the data flowing through the cache**

  Once a cacheable object is invoked, dynamic cache creates a cache entry for it that contains the output of the actions that are performed and metadata, such as time to live, sharing policy, and so on. Entries are distinguished by a unique ID string that is based on the rules specified in the `cachespec.xml` file for the particular object name. Objects with the same name might generate multiple cache IDs for different invocations, based on request parameters and attributes for each invocation. You can view of all the cache entries that are in the cache instance, based on the unique ID. You can also view the group of cache entries that share a common name (also known as template). Cache entries can also be grouped together by a dependency ID, which is used to invalidate the entire group of entries dependent on a common entity. Therefore, cache monitor also provides a view of the group of cache entries that share a common dependency ID.

  For each entry, cache monitor also displays metadata, such as time to live, priority and sharing-policy, and provides a view of the output that has been cached. This helps the customer to verify which pages have been cached, that the pages have been cached in the correct cache instance with the right attributes such as time to live, priority, and that the pages have the right content.

- **Monitor the data in the edge cache**

  Dynamic cache provides support to recognize the presence of an Edge Side Include (ESI) processor and to generate ESI include tags and appropriate cache policies for edge cacheable fragments. With the ESI processor, you can cache whole pages, as well as fragments, providing a higher cache hit ratio. There can be multiple ESI processors running on multiple hosts configured for caching.

  You can view a list of all ESI processes and their hosts that are enabled for caching. Select a host or a processor, and view its edge cache statistics and the current cache entries.

- **View the data offloaded to the disk**

  By default, when the number of cache entries reaches the configured limit for a given server, cache entries are removed, 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.

Edge cache statistics:

Cache monitor provides a view of the edge cache statistics.

The following statistics are available:

- **ESI Processors**. Number of processes configured as edge caches.
- **Number of Edge Cached Entries**. Number of entries currently cached on all edge servers and processes.
- **Cache Hits**. Number of requests that match entries on edge servers.
- **Cache Misses By URL.** A cache policy does not exist on the edge server, for the requested template.

  Note:

  - The initial ESI request for a template that has a cache policy on WebSphere Application Server results in a miss.
  - Every request for a template that does not have a cache policy on WebSphere Application Server results in a miss by URL on the edge server.
- **Cache Misses By Cache ID**. The policy for the requested template exists on the edge server, and a cache ID is created, based on the ID rules and the request attributes, but the cache entry for this ID does not exist.

  Note: If the policy exists on the edge server for the requested template, but a cache ID match is not found, based on the ID rules and the request attributes, it is not treated as a cache miss.
- **Cache Timeouts**. Number of entries removed from the edge cache, based on the timeout value.
- **Evictions**. Number of entries removed from the edge cache, due to invalidations received from WebSphere Application Server.

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″ in the information center.

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 1153 for more information.

1. Enable servlet caching. See ″Configuring servlet caching″ 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 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. 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 setting for a an environment without clustering is **Not Shared**. When enabling replication, the default value is **Push only**.

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″ and ″Configuring servlet caching″ in the information center.

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 1901 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 18. 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 19. 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 19. 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 1969. 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

Null value semantics: The following describe the semantics of NULL values:

- Comparison or arithmetic operations with an unknown (NULL) value yield an unknown value
- In a Java 2 platform, Enterprise Edition (J2EE) version 1.3 application, a path expression uses an outer-join semantic where a NULL field or cmr value evaluates to NULL. In J2EE version 1.4, the path expression uses an inner-join semantic.
- The IS NULL and IS NOT NULL operators can be applied to path expressions and return TRUE or FALSE. Boolean operators AND, OR and NOT use three valued logic.

AND	True	False	Unknown
True	True	False	Unknown
False	False	False	False
Unknown	Unknown	False	Unknown

OR	True	False	Unknown
True	True	True	True
False	True	False	Unknown
Unknown	True	Unknown	Unknown

	NOT
True	False
False	True
Unknown	Unknown

Example: Null value semantics

```
select object(e) from EmpBean where e.salary > 10  and e.dept.budget > 100
```

If salary is NULL the evaluation of `e.salary > 10` returns unknown and the employee object is not returned. If the cmr field dept or budget is NULL evaluation of `e.dept.budget > 100` returns unknown and the employee object is not returned.

```
select object(e) from EmpBean where e.dept.budget is null
```

In J2EE 1.3 if dept or budget is NULL evaluation of `e.dept.budget` is null returns TRUE and the employee object is returned. In J2EE 1.4 the employee object is returned only if budget is NULL.

```
select object(e) from EmpBean e ,  in (e.dept.emps) e1 where e1.salary > 10
```

If dept is NULL, then the multivalued path expression `e.dept.emps` results in an empty collection (not a collection that contains a NULL value). An employee with a null dept value will not be returned.

```
select object(e) from EmpBean e where  e.dept.emps is empty
```

If dept is NULL the evaluation of the predicate in unknown and the employee object is not returned.

```
select object(e) from EmpBean e , EmpBean e1  where e member of  e1.dept.emps
```

If dept is NULL evaluation of the member of predicate returns unknown and the employee is not returned.

Date time arithmetic and comparisons: DATE, TIME and TIMESTAMP values may be compared with another value of the same type. Comparisons are chronological. Date time values can also be incremented, decremented, and subtracted.

If the datastore is DB2, then DB2 string representation of DATE, TIME and TIMESTAMP types can also be used. A string representation of a date or time can use ISO, USA, EUR or JIS format. A string representation of a timestamp uses ISO format.

Format	Date format	Date examples	Time format	Time examples
ISO	yyyy-mm-dd	1987-02-24 1987-2-24	hh.mm.ss	13.50.00 13.50
USA	mm/dd/yyyy	2/24/1987	hh:mm AM or PM	1:50 pm 02:10 AM
EUR	dd.mm.yyyy	24.02.1987 24.2.1987	hh.mm.ss	13.50.00 13.55
JIS	yyyy-mm-dd	1987-02-24	hh:mm:ss	13:50 13:50:05

Example 1: Date time arithmetic comparisons

```
e.hiredate > '1990-02-24'
```

The timestamp of February 24th, 1990 1:50 pm can be represented as follows:

```
'1990-02-24-13.50.00.000000'  or
'1990-02-24-13.50.00'
```

If the datastore is DB2, DB2 decimal durations can be used in expressions and comparisons. A date duration is a decimal(8,0) number that represents the difference between two dates in the format YYYYMMDD. A time duration is a decimal(6,0) number that represents the difference between two time values as HHMMSS. A timestamp duration is a decimal(20,6) number representing the differences between two timestamp values as YYYYMMDDHHMMSS.ZZZZZZ (ZZZZZZ is the number of microseconds and is to the right of the decimal point) .

Two date values (or time values or timestamp values) can be subtracted to yield a duration. If the second operand is greater than the first the duration is a negative decimal number. A duration can be added or subtracted from a datetime value to yield a new datetime value.

Example 2: Date time arithmetic comparisons

`DATE('3/15/2000') - '12/31/1999'` results in a decimal number 215 which is a duration of 0 years, 2 months and 15 days.

Durations are really decimal numbers and can be used in arithmetic expressions and comparisons.

`( DATE('3/15/2000') - '12/31/1999' ) + 14 > 215` evaluates to TRUE.

`DATE('12/31/1999') + DECIMAL(215,8,0)` results in a date value 3/15/2000.

`TIME('11:02:26') - '00:32:56'` results in a decimal number 102930 which is a time duration of 10 hours, 29 minutes and 30 seconds.

`TIME('00:32:56') + DECIMAL(102930,6,0)` results in a time value of 11:02:26.

`TIME('00:00:59') + DECIMAL(240000,6,0)` results in a time value of 00:00:59.

`e.hiredate + DECIMAL(500,8,0) > '2000-10-01'` means compare the hiredate plus 5 months to the date 10/01/2000.

Basic predicates: Basic predicates can be of two forms

```
expression-1  comparison-operator  expression-2
expression-3  comparison-operator ( subselect )
```

The subselect must not return more than one value and the subselect can not return a type of an EJB reference. Boolean types and reference types only support = and <> comparisons.

Example: Basic predicates

```
d.name='Java Development'
e.salary > 20000
e.salary > ( select avg(e.salary) from EmpBean e)
```

Quantified predicates: A quantified predicate compares a value with a set of values produced by a subselect.

```
expression   comparison-operator   SOME  |  ANY  |   ALL    ( subselect )
```

The expression must not evaluate to a reference type.

When SOME or ANY is specified the result of the predicate is as follows:
- TRUE if the comparison is true for at least one value returned by the subselect.
- FALSE if the subselect is empty or if the comparison is false for every value returned by the subselect.
- UNKNOWN if the comparison is not true for all of the values returned by the subselect and at least one of the comparisons is unknown because of a null value.

When ALL is specified the result of the predicate is as follows:
- TRUE if the subselect returns empty or if the comparison is true to every value returned by the subselect.
- FALSE if the comparison is false for at least one value returned by the subselect.
- UNKNOWN if the comparison is not false for all values returned by the subselect and at least one comparison is unknown because of a null value.

BETWEEN predicate: The BETWEEN predicate determines whether a given value lies between two other given values.

```
expression  [NOT]  BETWEEN expression-2  AND expression-3
```

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
```

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 Javadoc (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 ejbbean.ear
   b. map the ejbbean.ear file to the server that the queried CMP bean is installed on.
5. Compile and run your client program with the file **qryclient.jar** in the classpath.

Example: Dynamic query remote interface:

When you run a dynamic EJB query using the remote interface, you are calling the executeQuery method on the Query interface. The executeQuery method has a transaction attribute of REQUIRED for this interface; therefore you do not need to explicitly establish a transaction context for the query to run.

Begin with the following import statements:

```
import com.ibm.websphere.ejbquery.QueryHome;
import com.ibm.websphere.ejbquery.Query;
import com.ibm.websphere.ejbquery.QueryIterator;
import com.ibm.websphere.ejbquery.IQueryTuple;
import com.ibm.websphere.ejbquery.QueryException;
```

Next, write your query statement in the form of a string, as in the following example that retrieves the names and ejb-references for underpaid employees:

```
String query =
"select e.name as name , object(e) as emp from EmpBean e where e.salary < 50000";
```

Create a Query object by obtaining a reference from the QueryHome class. (This class defines the executeQuery method.) Note that for the sake of simplicity, the following example uses the dynamic query JNDI name for the Query object:

```
InitialContext ic =  new InitialContext();

Object obj =  ic.lookup("com/ibm/websphere/ejbquery/Query");

QueryHome  qh =
 ( QueryHome) javax.rmi.PortableRemoteObject.narrow( obj, QueryHome.class );
Query qb = qh.create();
```

You then must specify a maximum size for the query result set, which is defined in the QueryIterator object. (See *Class QueryIterator* in Javadoc 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 Javadoc 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 Javadoc for more details.) Keep in mind that the iterator loses validity at the end of the transaction; you must use the iterator in the same transaction scope as the executeQuery call.

```
userTransaction.begin();
QueryLocalIterator it = qb.executeQuery(query, null, null);
while (it.hasNext() ) {
 IQueryTuple tuple = (IQueryTuple) it.next();
 System.out.print( it.getFieldName(1) );
 String s = (String) tuple.getObject(1);
 System.out.println( s);
 System.out.println( it.getFieldName(2) );
 EmpLocal e = ( EmpLocal ) tuple.getObject(2);
 System.out.println( e.getPrimaryKey().toString());
}
userTransaction.commit();
```

In most situations, the QueryLocalIterator object is *demand-driven*. That is, it causes data to be returned incrementally: for each record retrieval from the database, the next() method must be called on the iterator. (Situations can exist in which the iterator is not demand-driven. For more information, consult the ″Local query interfaces″ subsection of the Dynamic query performance considerations topic.)

Because the full query result set materializes incrementally in the application server memory, you can easily control its size. During a test run, for example, you may decide that return of only a few tuples of the query result is necessary. In that case you should use a call of the close() method on the QueryLocalIterator object to close the query loop. Doing so frees SQL resources that the iterator uses. Otherwise, these resources are not freed until the full result set accumulates in memory, or the transaction ends.

Dynamic query performance considerations: **General performance considerations**

Use of the following elements in your dynamic query can diminish application performance somewhat:

- Datatype converters and Java methods

  Why: In general, query operations and predicates are translated into SQL so that the database server can perform them. If your query includes datatype converters (for EJB to RDB mapping, for example) or Java methods, however, the associated predicates and operations of your query must be performed in the memory of the application server.
- EJB methods and criteria that call for the return of EJB references

  Why: Queries that incorporate these elements trigger full activation of EJBs in the memory of the application server. (Returning a list of CMP fields from a query does not cause an EJB to be activated.)

When assessing application performance, you should also be aware that dynamic queries share connections with the persistence manager. Consequently, an application that includes a mixture of finder methods, CMR navigation, and dynamic queries relies on a single shared connection between the persistence manager and the dynamic query service to perform these tasks.

Limiting the return collection size

- **Remote interface queries**: The QueryIterator class of the remote interface mandates that all of your query results materialize in application server memory over the course of one method call. The SQL cursor(s) used to run the EJB query are closed upon completion of that call. Because this requirement poses a high risk for creating bottlenecks within the database server, you need to limit the size of any potentially large result collections.
- **Local interface queries**: In most situations, the QueryLocalIterator object behaves as a wrapper around an SQL cursor. It is *demand-driven*; it causes data to be returned incrementally. For each record retrieval from the database, the next() method must be called on the iterator.

  Use of certain operations in local interface queries, however, overrides the demand-driven behavior. In these cases, the query results fully materialize in memory just as do the result collections of remote interface queries. An example of such a case is:

  ```
  select  e.myBusinessMethod( ) from EmpBean e
   where e.salary < 50000 order by 1 desc
  ```

  This query requires performance of an EJB method to produce the final result collection. Consequently, the full dataset from the database must be returned in one collection to application server memory, where the EJB method can be run on the dataset in its entirety. For that reason, local interface query operations that invoke EJB methods are generally not demand-driven. You cannot control the return collection size for such queries.

  Because they *are* demand-driven, all other local interface queries allow you to control the size of return collections. You can use a call of the close() method on the QueryLocalIterator object to close the query loop after the desired number of return values has been fetched from the datastore. Otherwise, the SQL cursor(s) used to run the EJB query are not closed until the full result set accumulates in memory, or the transaction ends.

Access intent implications for dynamic query: WebSphere Application Server gives you the option to set access intent policies for your entity enterprise beans as a way of managing their transfer of data with the underlying datastore. An access intent policy controls the isolation level used on the data source connection, as well as the database locks used during data retrieval. By manipulating these elements, you can maximize the efficiency of your application's data flow. To learn more, begin with the topics "Access intent policies" on page 822 and "Concurrency control" on page 822.

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″ in the information center for details.

Dynamic query API: prepareQuery() and executePlan() methods:

Use these methods to more efficiently allocate the overhead associated with dynamic query. They are equivalent in function to the prepareStatement() and executeQuery() methods of the JDBC API.

To perform a dynamic EJB query, the application server must parse the query string into SQL at run time. You can, of course, eliminate run-time overhead by choosing to perform a standard EJB query instead of a dynamic query. Sometimes referred to as *deployment queries*, standard queries are parsed and built at deployment, then performed by a finder or select method.

Another option is to write code that redistributes dynamic query overhead for better application performance. Begin by calling the prepareQuery() method in place of the executeQuery() method. The prepareQuery() method parses and translates your query, and returns a string called a *query plan*. The plan contains the SQL statement produced by parsing and translation, as well as other information needed by the dynamic query API. Save this string in your application and call the executePlan() method with the string to run your query. (You also might want to use the prepareQuery() method simply to see the SQL translation product; just call the method and display the return value.)

Pass the parameters of your query as an array of type Object on the prepareQuery() and the executePlan() method calls. Ensure that you pass appropriate data types, because the application server validates your query according to parameter type (rather than actual values) when it processes the prepareQuery() method call.

Example code

Note: In the example code that follows, the first executePlan() method call substitutes `parms[0]` for `?1`. Hence the first query performed is functionally equivalent to the following query statement:

```
 select e.name as name, object(e) as emp from EmpBean e where e.salary < 50000
```

The second call runs a query that is functionally equivalent to this statement:

```
 select e.name as name, object(e) as emp from EmpBean e where e.salary < 60000
```

The example:

```
String query =
"select e.name as name , object(e) as emp from EmpBean e where e.salary < ?1";
QueryIterator it = null;
Integer[] parms = new Integer[1];
parms[0] = new Integer(0);
```

In the call to `prepareQuery()`, pass any Integer value. Doing so defines `?1` as an Integer type, as in the following:

```
String queryPlan= qb.prepareQuery(query, parms, null );

   parms[0] = new Integer(50000);
```

Next you run the query with a real value of `Integer(50000)` for `?1`:

```
select e.name as name, object(e) as emp from EmpBean e
  where e.salary < 50000it = qb.executePlan( queryPlan, parms, 0, 99);

parms[0] = new Integer(60000);
```

Run the query again with a different value of `Integer(60000)` for `?1`:

```
it = qb.executePlan( queryPlan, parms, 0, 99);
```

Comparison of the dynamic and deployment EJB query services: You can use the dynamic query service to build and execute queries against entity beans constructed dynamically at 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 1981.
- 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 1982.

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 1982.

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. See ″Assembling applications″ and ″Assembly tools″ in the information center.
5. Prepare the localizable-text package for deployment with your localized application. See ″Prepare 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.
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 for additional information. 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 1978.

Java 2 Platform, Enterprise Edition (J2EE) provides support for application components that run on computers with differing endian architecture and code sets. It does not provide dedicated support for application components that run on computers with different locales or time zones.

The internationalization service addresses the challenges posed by locale and time zone mismatch without incurring the limitations of conventional techniques. The service systematically manages the distribution of internationalization contexts across the various components of EJB applications, including client applications, enterprise beans, and servlets.

The service works by associating an internationalization context with every service request within an application. When a client-side component calls a business method, the internationalization service interposes by obtaining the internationalization context associated with the current client-side process and by attaching that context to the outgoing request. On the server side, the internationalization service again interposes by detaching the context from the incoming request and associating it with the server-side process on which the business method will run, effectively scoping the context to the business method. For HTTP requests, the caller context is constructed from the HTTP attributes and default values. The service propagates internationalization context on subsequent business method invocations in the same manner, which distributes the context of the originating request over the entire chain of business method invocations.

This basic operation of scoping and propagation is defined precisely by internationalization context management policies. Internationalization policies specify whether an application component or its hosting J2EE container are to manage internationalization context. For container-managed components, the policy indicates which internationalization context the container scopes to invocations on that component. Server components configured to manage internationalization context, as well as EJB clients, must use the internationalization context API to manage the internationalization context elements scoped to their invocations.

Every application component has a default policy, which can be overridden and tailored for servlets and enterprise beans at assembly time.

At run time, application components can use the internationalization context API to get any element of the internationalization contexts scoped to an invocation. To programmatically access context elements, application components first resolve an internationalization context API reference, then call the appropriate API method to access the various context elements, such as the caller locale or the invocation time zone. These elements can be used in calls to Java 2 SDK internationalization API methods; for example, to perform localizations such as formatting messages, configuring dates, or comparing strings.

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.

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.
2. Click **Servers** > **Application servers** > *server_name* > **Container services** > **Internationalization service**.
3. Enable the internationalization service.
   a. If not already selected, select the **Enable service at server startup** check box.
   b. Click **OK**.

When you select the **Enable service at server startup** setting, the application server automatically initializes and starts the internationalization service whenever the server starts. If you change this setting, be sure to restart the application server for the new setting to take effect.

To disable the service, clear the **Enable service at server startup** check box. In this case, the internationalization service is initialized but not started when the application server starts.

Administration through scripting

Alternatively, the internationalization service can be enabled from the command line by using the wsadmin tool. Start the wsadmin tool and enter the following commands:

```
set x [$AdminConfig list I18NService]
$AdminConfig modify $x { { enable true } }
$AdminConfig save
exit
```

If you enable or disable the internationalization service, be sure to stop and then restart the application server for the new setting to take effect.

Enabling the internationalization service for EJB clients:

By default, the internationalization service is 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.

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**.

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 " on page 1986
- "The service is not started" on page 1986
- "Invalid context element" on page 1987
- "Missing context element" on page 1987
- "Invalid policy" on page 1987
- "Missing policy" on page 1987

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 the *Troubleshooting and support* PDF for information on 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 1983.

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″ in the information center.

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

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.

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.

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.

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.

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.

Pool Class Name:

The fully qualified class name of the objects that are stored in the object pool.

Data type	String

Pool Impl Class Name:

The fully qualified class name of the CustomObjectPool implementation class for this object pool.

Data type	String

Object pool service settings:

Use this page to enable or disable the object pool service, which manages object pool resources used by the server.

To view this administrative console page, click **Servers > Application Servers >** *server_name* > Container services > **Object Pool Service**.

Enable service at server startup:

Specifies whether the server attempts to start the object pool service.

Default	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

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 Java Beans 2.1 documentation for more details.

Use the following table to determine which persistent timer service is best for you:

Schedulers	EJB timers

Run stateless session EJB components and sends JMS messages	Run all EJB types except for stateful session beans
Persistent, transactional and highly available.	Persistent, transactional and highly available.
Tasks guaranteed to run only once	Timers guaranteed to run only once, if the timer EJB uses a container-managed global transaction
Run repeating tasks using any calculation rules	Run repeating tasks using a repeating interval defined in milliseconds
Uses a modified fixed-delay time calculation to determine repeating intervals (next run time based on the start-time of the previous task)	Uses a fixed-rate time calculation to determine repeating intervals (time of the next task is based on the original scheduled time).
Programmatic task monitoring capability with the use of the NotificationSink stateless session EJB	No programmatic timer monitoring
Abort late or time-sensitive tasks from running	Abort late or time-sensitive tasks from running (achieved through manual detection within the javax.ejb.TimedObject implementation).
Manage any task lifecycle (find, suspend, resume, cancel and purge tasks programmatically and through Java Management Extensions (JMX)).	Find and cancel its timers programmatically. Administrators find and cancel timers using a command-line utility.
Store a limited amount of text with the data, like a **Name** (arbitrary data stored externally.)	Store arbitrary data with a timer

This task demonstrates how to manage, develop and interoperate with schedulers and subsequent tasks.

1. Manage the scheduler service. This article includes instructions for creating and configuring schedulers, creating and configuring a database for schedulers and administering schedulers.
2. Develop and schedule tasks. This article includes instructions for developing various types of tasks, receiving notifications from a task, submitting tasks to a scheduler, and managing tasks.

   Note: Creating and manipulating scheduled tasks through the Scheduler API interface is only supported from within the Enterprise Java Beans (EJB) container or Web container (JavaServer Pages or servlets). Looking up and using a configured scheduler from a Java 2 Platform Enterprise Edition (J2EE) application client container is not supported.
3. Interoperate with schedulers. This article explains how to manage scheduler in a clustered environment with mixed WebSphere Application Server product versions and mixed platforms.

Scheduler daemon:

A scheduler daemon is a background thread that searches for tasks to run in the database.

A scheduler daemon is started for each scheduler defined on each server. If `Scheduler 1` is configured on `server1`, then only one scheduler daemon runs on `server1` unless it is cloned. If `Scheduler 1` is defined at the node scope level, then the scheduler will run on each server within that node.

The poll interval determines the frequency at which the persistent store is queried. By default, this value is set to 30 seconds. When a task is found that is scheduled to run within the current poll interval, an asynchronous beans alarm is set. The task then runs as close to this time as possible using an alarm thread from the scheduler's associated work manager. Thus, the number of alarm threads configured on the work manager determines how many concurrent tasks are executed. No tasks are lost. If we reach this limit, then new tasks are simply queued to be executed when an alarm thread becomes available. The actual firing time is dictated by server load and availability of free threads in the alarm thread pool of the associated work manager.

Scheduler daemons in a cluster

When multiple schedulers are configured to use the same tables (as is the case in a clustered environment), any of the daemons can find a task and set the alarm in its Java virtual machine (JVM). The task is executed in the virtual machine where the scheduler daemon first runs, until the daemon is stopped and another daemon starts. If an application on `server1` schedules a task to run and `server2` was started before `server1`, then the task runs on `server2`.

Example: Stopping and starting scheduler daemons using Java Management Extensions API:

This example JACL script can be invoked using the wsadmin scripting tool. It will attempt to stop and start a Scheduler daemon.

```
# Example JACL Script to restart a Scheduler Daemon

set schedJNDIName sched/MyScheduler

# Find the WASScheduler MBean
regsub -all {/} $schedJNDIName "." schedJNDIName
set mbeanName Scheduler_$schedJNDIName
puts "Looking up Scheduler MBean $mbeanName"
set sched [$AdminControl queryNames WebSphere:*,type=WASScheduler,name=$mbeanName]

# Invoke the stopDaemon operation.
puts "Stopping the daemon..."
$AdminControl invoke $sched stopDaemon
puts "The daemon has stopped."

# Invoke the startDaemon operation.
puts "Starting the daemon..."
$AdminControl invoke $sched startDaemon 0
puts "The daemon has started."
```

Example: Dynamically changing scheduler daemon poll intervals using Java Management Extensions API:

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."
```

Managing schedulers

Schedulers are configured using the administrative console, configuration service or scripting and are available to all servers on which a scheduler is visible. You can create multiple schedulers within a single server, cluster, node or cell. Each configured scheduler is an independent task scheduling engine that has a unique Java Naming and Directory Interface (JNDI) name, persistent storage device and daemon.

1. Configure schedulers.
2. Create the database for schedulers.

Configuring schedulers:

Before your application can make use of the scheduler service, you must configure a scheduler using the administrative console, configuration service or scripting. Conceptually, a scheduler is similar to a data source in that you must specify various configuration attributes, including a JNDI name where the instance is bound. Once defined, an application using the Scheduler API or WASScheduler MBean can look up the scheduler object and call various methods to manage tasks.

The scheduler service is always enabled. In previous versions of the product, the scheduler service could be disabled using the administrative console or configuration service. Scheduler service configuration objects are present in the configuration service, but the enabled attribute is ignored.

To achieve high availability, you can configure a duplicate scheduler on each server in a cluster, or create a scheduler at the cluster scope. For example, each server that contains a scheduler with the JNDI name `sched/MyScheduler`, with the same database configuration parameters (data source and table prefix) behaves as a single clustered scheduler. Each server in the scheduler cluster has a running scheduler instance, which increases the number of poll daemons and allows automatic failover. For more information on creating clusters for high availability, see the article, ″WebSphere Enterprise Scheduler planning and administration guide.″

Typically, create schedulers at the server or cluster scope. Scheduler poll daemons run in each server within the configured scope, which means that if you create a scheduler at the node or cell scope, the scheduler poll daemon can attempt to run tasks on any of the servers in the node or cell. If applications are not mapped uniformly over each server in that scope, the scheduler might not run tasks correctly. Since applications are mapped to servers and clusters, there is less chance for error and less competition between daemons to run tasks.

Depending on your preferred method of configuration, select one of the following steps to configure schedulers.

1. Configuring schedulers using the administrative console.
2. Configuring schedulers using Java Management Extensions API (JMX).

A scheduler is configured and ready to use.

Configuring 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″ in the information center.
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.

Schedulers collection:

Use this page to manage scheduler configurations.

To view this administrative console page, click **Resources > Schedulers**.

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*.

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:

Datasource 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.

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

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 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.

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.

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.

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.

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.

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.

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.

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.

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.

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″ in the information center. To see the exact schema definition such as field sizes and types, see ″Creating scheduler tables using DDL files″ in the information center. 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.

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.

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.

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 in the information center 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 ""
}
```

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 in the information center 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 ""
}
```

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 in the information center 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 ""
}
```

Creating scheduler tables using DDL files:

Your database system must be installed and available.

The scheduler requires a database, a JDBC provider, and a data source.

Complete the following steps to create scheduler tables using DDL files.

1. Verify that your database is created. See the topic ″Creating scheduler databases.″
2. Create the database tables according to the instructions for your database system.
   - Creating Cloudscape tables for schedulers.
   - Creating DB2 tables for schedulers.
   - Creating DB2 tables for z/OS for schedulers.
   - Creating Informix tables for schedulers.
   - Creating Microsoft SQL Server tables for schedulers.
   - Creating Oracle tables for schedulers.
   - Creating Sybase tables for schedulers.

Creating Cloudscape tables for schedulers:

This 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.

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.

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.

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.

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.

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.

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.

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. All priorities must be positive integers. Beans are stopped in the opposite order to their start priority.

View the startup beans service settings.

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 application start time and application 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**.

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.

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. Refer to the information center or to the *Administering applications and their environment* PDF for details.
2. Managing work areas. The work area service is managed using the administrative console. . Refer to the information center or to the *Administering applications and their environment* PDF for details.

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.

Work area property modes: The information in a work area consists of a set of properties; a property consists of a key-value-mode triple. The key-value pair represents the information contained in the property; the key is a name by which the associated value is retrieved. The mode determines whether the property can be removed or modified.

Property modes

There are four possible mode values for properties, as shown in the following code example:

Code example: The PropertyModeType definition

```
public final class PropertyModeType {
   public static final PropertyModeType normal;
   public static final PropertyModeType read_only;
   public static final PropertyModeType fixed_normal;
   public static final PropertyModeType fixed_readonly;
};
```

A property’s mode determines three things:
- Whether the value associated with the key can be modified
- Whether the property can be deleted
- Whether the mode associated with the key-value pair can be modified

The two read-only modes forbid changes to the information in the property; the two fixed modes forbid deletion of the property.

The work area service does not provide methods specifically for the purpose of modifying the value of a key or the mode associated with a property. To change information in a property, applications simply rewrite the information in the property; this has the same effect as updating the information in the property. The mode of a property governs the changes that can be made. Modifying key-value pairs describes the restrictions each mode places on modifying the value and deleting the property. Changing modes describes the restrictions on changing the mode.

Changing modes

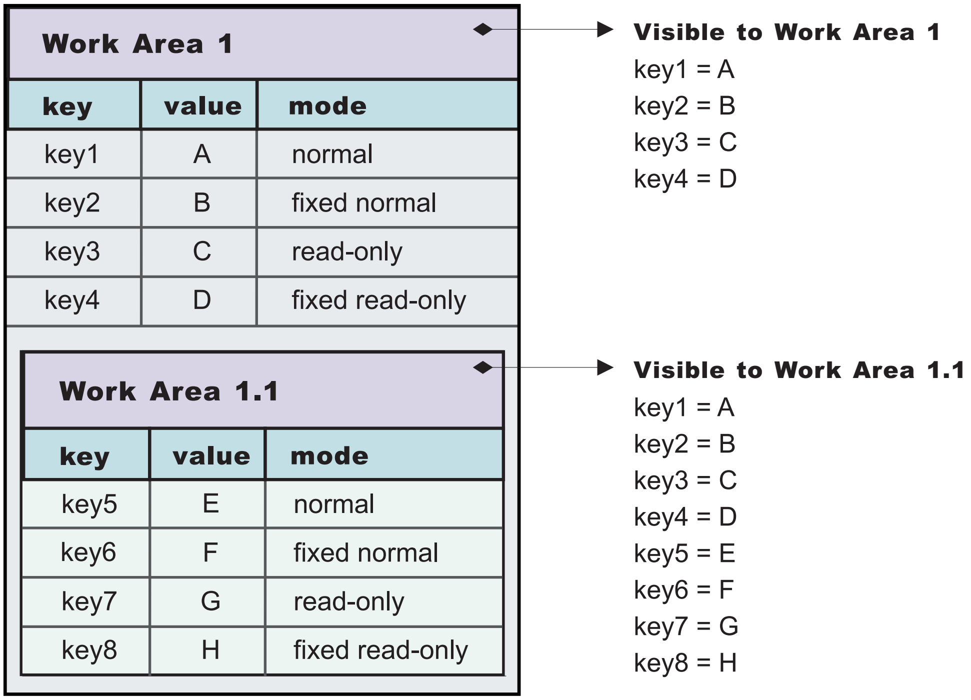
The mode associated with a property can be changed only according to the restrictions of the original mode. The read-only and fixed read-only properties do not permit modification of the value or the mode. The fixed normal and fixed read-only modes do not allow the property to be deleted. This set of restrictions leads to the following permissible ways to change the mode of a property within the lifetime of a work area:

- If the current mode is normal, it can be changed to any of the other three modes: fixed normal, read-only, fixed read-only.
- If the current mode is fixed normal, it can be changed only to fixed read-only.
- If the current mode is read-only, it can be changed only by deleting the property and re-creating it with the desired mode.
- If the current mode is fixed read-only, it cannot be changed.
- If the current mode is not normal, it cannot be changed to normal. If a property is set as fixed normal and then reset as normal, the value is updated but the mode remains fixed normal. If a property is set as fixed normal and then reset as either read-only or fixed read-only, the value is updated and the mode is changed to fixed read-only.

Note: The key, value, and mode of any property can be effectively changed by terminating (completing) the work area in which the property was created and creating a new work area. Applications can then insert new properties into the work area. This is not precisely the same as changing the value in the original work area, but some applications can use it as an equivalent mechanism.

Nested work areas: Applications can nest work areas. When an application creates a work area, a work area context is associated with the creating thread. If the application thread creates another work area, the new work area is nested within the existing work area and becomes the current work area. Nested work areas allow applications to define and scope properties for specific tasks without having to make them available to all parts of the application. All properties defined in the original, enclosing work area are visible to the nested work area. The application can set additional properties within the nested work area that are not part of the enclosing work area.

An application working with a nested work area does not actually see the nesting of enclosing work areas. The current work area appears as a flat set of properties that includes those from enclosing work areas. In the figure below, the enclosing work area holds several properties and the nested work area holds additional properties. From the outermost work area, the properties set in the nested work area are not visible. From the nested work area, the properties in both work areas are visible.



Work Area 1

key	value	mode
key1	A	normal
key2	B	fixed normal
key3	C	read-only
key4	D	fixed read-only

Work Area 1.1

key	value	mode
key5	E	normal
key6	F	fixed normal
key7	G	read-only
key8	H	fixed read-only

Visible to Work Area 1
key1 = A
key2 = B
key3 = C
key4 = D

Visible to Work Area 1.1
key1 = A
key2 = B
key3 = C
key4 = D
key5 = E
key6 = F
key7 = G
key8 = H

Figure 11. 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.

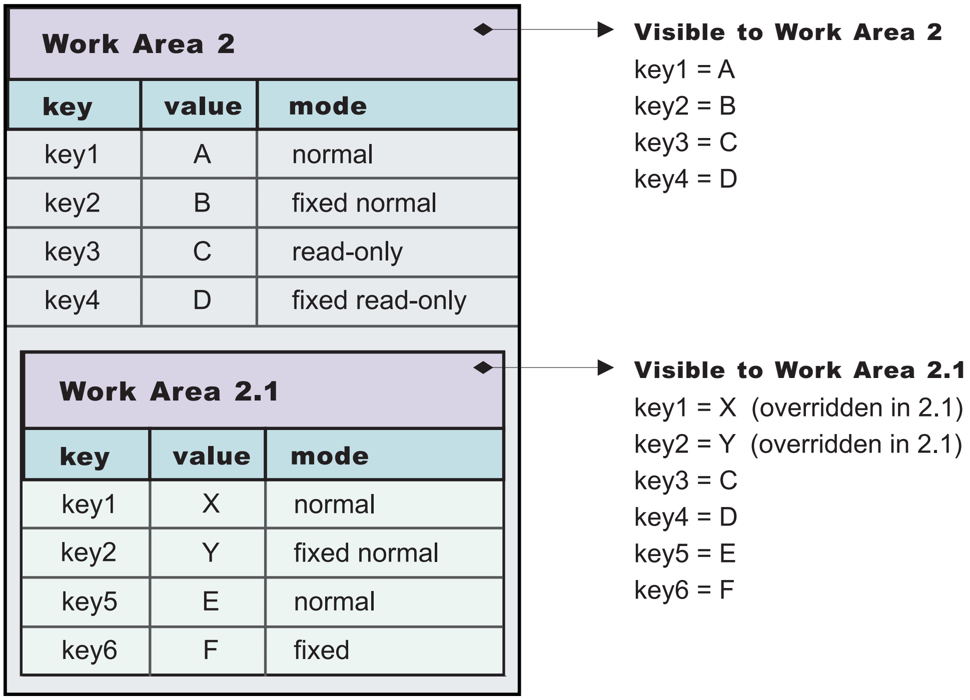
The mode associated with a property when it is created determines whether nested work areas can override the property. From the perspective of a nested work area, the property modes used in enclosing work areas can be grouped as follows:

- 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.



Work Area 2

key	value	mode
key1	A	normal
key2	B	fixed normal
key3	C	read-only
key4	D	fixed read-only

Work Area 2.1

key	value	mode
key1	X	normal
key2	Y	fixed normal
key5	E	normal
key6	F	fixed

Visible to Work Area 2
key1 = A
key2 = B
key3 = C
key4 = D

Visible to Work Area 2.1
key1 = X (overridden in 2.1)
key2 = Y (overridden in 2.1)
key3 = C
key4 = D
key5 = E
key6 = F

Figure 12. Redefining existing properties in nested work areas

Distributed work areas: The propagation of work area context operates differently depending on whether a work area partition is defined as bidirectional or not. In either case all work area context propagates to a target object on a remote invocation. However, whether the context propagates from a target object back to the originator depends on whether a partition is defined as bidirectional.

Non-bidirectional work area partitions (UserWorkArea partition)

If a remote invocation is issued from a thread associated with a work area, a copy of the work area is automatically propagated to the target object, which can use or ignore the information in the work area as necessary. If the calling application has a nested work area associated with it, a copy of the nested work area and all its ancestors is propagated to the target. The target application can locally modify the information, as allowed by the property modes, by creating additional nested work areas; this information is propagated to any remote objects it invokes. However, no changes made to a nested work area on a target object are propagated back to the calling object. The caller's work area is unaffected by changes made in the remote method.

Bidirectional work area partitions

If a remote invocation is issued from a thread associated with a work area, a copy of the work area is automatically propagated to the target object, which can use or ignore the information in the work area as necessary. If the calling application has a nested work area associated with it, a copy of the nested work area and all its ancestors is propagated to the target. The target application can locally modify the information, as allowed by the property modes, this information is propagated to any remote objects it invokes. In a partition that is not defined as bidirectional, a target application must begin a nested work area before making changes to the imported work area. However, if a partition is defined as bidirectional, a target application need not begin a nested work area before operating on an imported work area. By not beginning a nested work area, any new context set into the work area, or any context changes made by the target application, is not only propagated on future remote invocations but is also propagated back to the originating application (that is, the one who initiated the remote invocation) thus allowing bidirectional propagation of work area context. If the target application does not want new or changed context to propagate back to the originating application, then the target application must begin a nested work area to scope the context to its process. However, the new or changed context in the nested work area propagates on any future remote invocation the target application may make.

WorkArea service: Special considerations: Developers who use work areas should consider the following issues that could potentially cause problems: interoperability between the EJB and CORBA programming models; and the use of work areas with Java's Abstract Windowing Toolkit.

EJB and CORBA interoperability

Although the WorkArea 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, but 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. If a J2EE application tries to retrieve CORBA-set properties that are non-serializable, or contain CORBA nulls or void references, the com.ibm.websphere.workarea.IncompatibleValue exception is raised.

Using work areas with Java's Abstract Windowing Toolkit (AWT)

Work areas must be used cautiously in applications that use Java's Abstract Windowing Toolkit (AWT). The AWT implementation is multithreaded, and work areas begun on one thread are not available on another. For example, if a program begins a work area in response to an AWT event, such as pressing a button, the work area might not be available to any other part of the application after the execution of the event completes.

Work area service performance considerations: The work area service is designed to address complex data passing patterns that can quickly grow beyond convenient maintenance. A *work area* is a note pad that is accessible to any client that is capable of looking up Java Naming Directory Interface (JNDI). After a work area is established, data can be placed there for future use in any subsequent method calls to both remote and local resources.

You can utilize a work area when a large number of methods require common information or if information is only needed by a method that is significantly further down the call graph. The former avoids the need for complex parameter passing models where the number of arguments passed becomes excessive and hard to maintain. You can improve application function by placing the information in a work area and subsequently accessing it independently in each method, eliminating the need to pass these parameters from method to method. The latter case also avoids unnecessary parameter passing and helps to improve performance by reducing the cost of marshalling and de-marshalling these parameters over the Object Request Broker (ORB) when they are only needed occasionally throughout the call graph.

When attempting to maximize performance by using a work area, cache the UserWorkArea partition that is retrieved from JNDI wherever it is accessed. You can reduce the time spent looking up information in JNDI by retrieving it once and keeping a reference for the future. JNDI lookup takes time and can be costly.

Additional caching mechanisms available to a user-defined partition are defined by the configuration property, ″Deferred Attribute Serialization″. This mechanism attempts to minimize the number of serialization and deserialization calls. See ″Work area partition service″ in the information center 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 2028, to access the UserWorkArea partition.
   - “Accessing a user defined work area partition” on page 2044, 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

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″ in the information center.

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″ in the information center.

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 2044. 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

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 2027 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

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 2028 or a reference to a user defined partition as defined in “Accessing a user defined work area partition” on page 2044. 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 2027 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.

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();
      ...
   }
}
```

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 2043for 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) {
      }
      ...
  }
}
```

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");
       ...
   }
}
```

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.

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.

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) {
      }
   }
}
```

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.

Refer to the information center or to the *Administering applications and their environment* PDF for details.

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.

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** .

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.

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 2040, for an example.

Work area partition service:

The work area partition service is an extension of the work area service that allows the creation of multiple custom work areas. The work area partition service is an optional service to users. Any user that currently uses the work area service and the UserWorkArea partition can continue using it in the same manner. The UserWorkArea partition is created automatically (if it has not been disabled) by the work area partition service. By allowing a user the option to create their own work area partition through the work area partition service, they can have more control over configuration and access to their partition.

Unlike the UserWorkArea partition, which is publicly known, work areas created by the work area partition service are accessible to, and known only by the creator. However, the work area partition service does not strictly enforce that a partition is accessed and/or operated on exclusively by the partition creator. There are no limitations should the creator want to publish their work area partition and make it publicly available by binding their partition reference in java naming or by other means. However, the work area partition service does try to hide a partition as much as possible should a user not want others to know about a certain partition. The work area partition service does not allow a person to determine, or query the names of all the partitions that have been created; however, it does not restrict the partitions from being accessed by users other than the creator of that partition. The context of a partition, such as the UserWorkArea partition or a user defined partition, is scoped to a single thread and is not accessible by multiple threads.

The work area partition reference that is returned to a user implements javax.naming.Referenceable, as well as com.ibm.websphere.UserWorkArea, therefore a user can bind their partition into a name to make their partition publicly available. An alternative to using Java naming to bind and access the partition is to use the work area partition manager interface. Anyone can access the work area partition manager interface; therefore, if a user wants to make their partition publicly available, they simply need to publish their partition name. Other users can then call the getWorkAreaPartition method on the work area partition manager interface with the published name.

The WorkAreaPartitionManager.createWorkAreaPartition method can only be used from a Java 2 Platform, Enterprise Edition (J2EE) client. To create a work area partition on the server side, one must use the administrative console. On the server side a work area partition must be created during server startup because each partition needs to be register with the appropriate Web and Enterprise JavaBeans (EJB) collaborators before the server has started. Custom work area partitions are created by the work area partition service and defined by the UserWorkArea interface.

The work area partition service also allows a user to configure partitions with additional properties that are not available on the UserWorkArea partition, such as bidirectional propagation of work area partition context and deferred attribute serialization. These properties are available as configuration properties when creating a partition. 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 2043 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″ in the information center for configuration parameters available to configure a partition.

```
import com.ibm.websphere.workarea.WorkAreaPartitionManager;
import com.ibm.websphere.workarea.UserWorkArea;
import com.ibm.websphere.workarea.PartitionAlreadyExistsException;
import com.ibm.websphere.workarea.NoSuchPartitionException;
import java.lang.IllegalAccessError;
import java.util.Properties;
import javax.naming.InitialContext;

//This sample demonstrates how to retrieve an instance of the
//WorkAreaPartitionManager implementation and how to use that
//instance to create a WorkArea partition and retrieve a partition.
//NOTE: Creating a partition in the way listed below is only available
//on a J2EE client.  To create a partition on the server use the
//WebSphere administrative console.  Retrieving a WorkArea
//partition is performed in the same way on both client and server.

public class Example {

     //The name of the partition to create/retrieve
     String partitionName = "myPartitionName";
     //The name in java naming the WorkAreaPartitionManager instance is bound to
     String jndiName = "java:comp/websphere/WorkAreaPartitionManager";

     //On a J2EE client a user would create a partition as follows:
     public UserWorkArea myCreate(){
         //Variable to hold our WorkAreaPartitionManager reference
         WorkAreaPartitionManager partitionManager = null;
         //Get an instance of the WorkAreaPartitionManager implementation
         try {
             InitialContext initialContext = new InitialContext();
             partitionManager = (WorkAreaPartitionManager) initialContext.lookup(jndiName);
         } catch (Exception e) {  }

         //Set the properties to configure our WorkArea partition
         Properties props = new Properties();
         props.put("maxSendSize","12345");
         props.put("maxReceiveSize","54321");
         props.put("Bidirectional","true");
      props.put("DeferredAttributeSerialization","true");

         //Variable used to hold the newly created WorkArea Partition
         UserWorkArea myPartition = null;
```

```
        try{
            //This is the way to create a partition on the J2EE client.  Use the
            //WebSphere Administrative Console to create a WorkArea Partition
            //on the server.
            myPartition = partitionManager.createWorkAreaPartition(partitionName,props);
        }
        catch (PartitionAlreadyExistsException e){  }
        catch (IllegalAccessException e){  }

        return myPartition;
    }

    //. . . .

    //In order to retrieve a WorkArea partition at some time later or
    //from some other class, do the following (from client or server):
    public UserWorkArea myGet(){
        //Variable to hold our WorkAreaPartitionManager reference
        WorkAreaPartitionManager partitionManager = null;
        //Get an instance of the WorkAreaPartitionManager implementation
        try {
            InitialContext initialContext = new InitialContext();
            partitionManager = (WorkAreaPartitionManager) initialContext.lookup(jndiName);
        } catch (Exception e) {  }

        //Variable used to hold the retrieved WorkArea partition
        UserWorkArea myPartition = null;
        try{
            myPartition = partitionManager.getWorkAreaPartition(partitionName);
        }catch(NoSuchPartitionException e){  }

        return myPartition;
    }
}
```

Work area partition collection:

Use this page to manage the work area service.

The work area partition service supports the definition of custom work area partitions.

To view this administrative console page, click **Servers > Application servers >** *server_name* **> Business process services> Work area partition service**.

Name:

Specifies the name of the work area partition that is used to retrieve the partition. This name must be unique.

Description:

Specifies the description of the work area partition.

Enable service at server startup:

Specifies whether the server attempts to start the specified service when the server starts.

Bidirectional:

Permits applications to modify the context of a work area that is imported by a J2EE request; modified properties are propagated back to the requestor environment. This option is disabled by default.

Maximum send size:

Specifies the maximum size of data that can be sent within a single work area. (0 = no limit; -1 = default)

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*.

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 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.

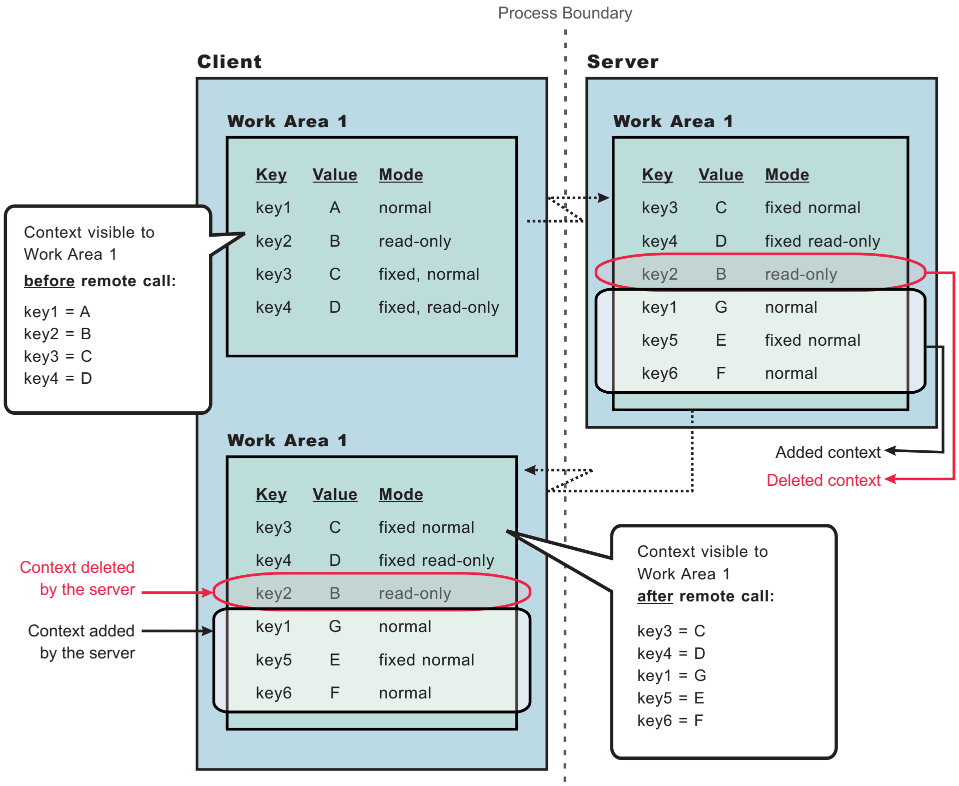


Figure 13. Figure 1

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.

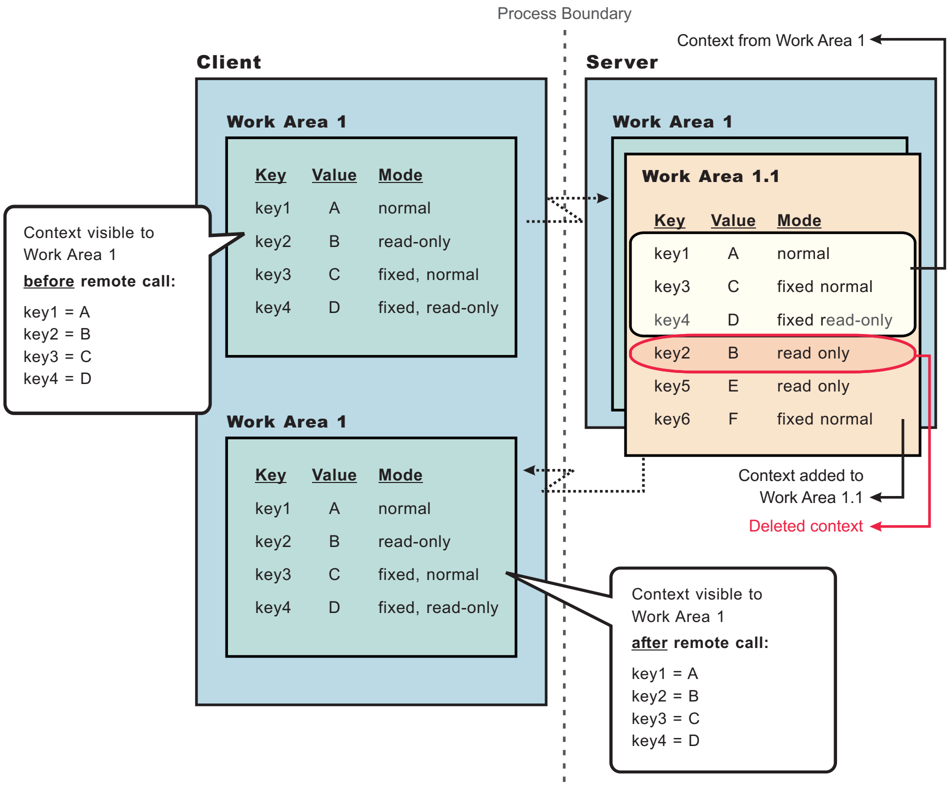


Figure 14. Figure 2

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″ in the information center.

Chapter 9. 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″ in the information center.

  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 ″Obtaining help from IBM″ in the information center.

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 C:/Documents and Settings/
  myUserName/Desktop/MyApp/myapp.ear"; exception information:
  com.ibm.bsf.BSFException: error while
  eval'ing Jacl expression: can't find method "installInteractive"
  with 3 argument(s) for class
  "com.ibm.ws.scripting.AdminAppClient"
  ```

  The file and path name are incorrectly specified. In this case, since the path included spaces, it was interpreted as multiple parameters by the wsadmin program.

  Enter the path of the `.ear` file correctly. In this case, by enclosing it in double quotes:

  ```
  $AdminApp installInteractive "C:\Documents
  and Settings\myUserName\Desktop\MyApps\myapp.ear"
  ```
- If the full text of the error is similar to:

  ```
  WASX7015E: Exception running command: "$AdminApp installInteractive c:\MyApps\myapp.ear ";
  exception information: com.ibm.ws.scripting.ScriptingException:  WASX7115E:
  Cannot read input file
  "c:\WebSphere\AppServer\bin\MyAppsmyapp.ear"
  ```

  The application path is incorrectly specified. In this case, you must use 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 **`$AdminApp installInteractive`** ***`filename`*** command to receive prompts for data sources during installation, or to provide them in a response file.
- Specify data sources as an option to the **`$AdminApp install`** command. For details on the syntax, see ″Installing applications with the wsadmin tool″ in the information center.

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″ in the information center 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:

- ″The server process does not start or starts with errors″ in the information center.
- “The application does not start or starts with errors” on page 2056.
- “A web resource does not display” on page 2057.
- ″Cannot access a data source″ in the information center.
- ″Cannot access an enterprise bean from a servlet, a JSP file, a stand-alone program, or another client″ in the information center.
- ″Cannot look up an object hosted by WebSphere Application Server from a servlet, JSP file, or other client″ in the information center.
- ″Access problems after enabling security″ in the information center.
- ″Errors after enabling security″ in the information center.
- ″Errors after configuring or enabling Secure Sockets Layer″ in the information center.
- ″Errors in messaging″ in the information center.
- ″Errors returned to a client sending a SOAP request″ in the information center.
- ″A client program does not work″ in the information center.
- ″Errors connecting to WebSphere MQ and creating WebSphere MQ queue connection factory″ in the information center.

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″ in the information center.

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
- File serving problems
- Graphics do not appear on the JavaServer Pages (JSP) file or servlet output
- SRVE0026E: [Servlet Error]-[Unable to compile class for JSP error on JSP]
- After modifying and saving a JSP file, the change does not show up in the browser (the old JSP file displays)
- Message similar to ″Message: /jspname.jsp(9,0) Include: Mandatory attribute page missing″ displays when trying to access JSP file
- The Java source generated from a JSP file is not retained in the temp directory (only the class file is found)
- The JSP batch compiler fails with the message ″Enterprise Application [application name you typed in] not found″
- There is a translation problem with non-English browser input
- Scroll bars do not appear around items in the browser window
- A ″Page cannot be displayed... server not found or DNS error″ error displays when attempting to browse a JavaServer Pages (JSP) file using Internet Explorer

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.
5. For further details on troubleshooting plug-in-related problems, see the topic ″Troubleshooting the HTTP plug-in component″ in the information center.

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. View the file serving property of the hosting Web module by browsing the source `.war` file in an assembly tool. See "Assembling applications" in the information center. If necessary, update the property and redeploy the module.
  2. 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. View the file serving property of the hosting Web module by browsing the source `.war` file in an assembly too.. If necessary, update the property and re-deploy the module.
  2. 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.
  3. After following the previous steps:
     - 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 Syst emErr.log file for a compiler error message, such as:

```
C:\WASROOT\temp\ ... test.war\_myJsp.java:14: \Duplicate variable declaration: int myInt was int myInt
int myInt = 122;
String myString = "number is 122";
static int myStaticInt=22;
int myInt=121;
      ^
```

Fix the problem in the JSP source file, save the source and request the JSP file again.

If this error occurs when trying to serve a JSP file that was copied from another system where it ran successfully, then there is something different about the new server environment that prevents the JSP file from running. Browse the text of the error for a statement like:

```
 Undefined variable or class name: MyClass
```

This error indicates that a supporting class or jar file is not copied to the target server, or is not on the class path. Find the MyClass.class file, and place it on the Web module WEB-INF/classes directory, or place its containing .jar file in the Web module WEB-INF/lib directory.

Verify that the URL used to access the resource is correct by completing the following steps:
- 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*.
  - For example, to access myCom.myServlet.class, located in c:\WebSphere\ApplicationServer\installedApps\ myEntApp.ear\myWebApp.war\WEB-INF\classes, and assuming the context root for the myWebApp module is ″myApp″, the URL would be `http://myhost.mydomain.com/myApp/servlet/myCom.MyServlet`.
- Serving servlets by class name must be enabled as a property of the Web module, and is enabled by default. File serving for HTML and image files must be enabled as a property of the Web application, in an assembly tool, or by setting the **fileServingEnabled** property to **true** in the **ibm-web-ext.xmi** file of the installed Web application and restarting the application.

Correct the URL in the ″from″ HTML file, servlet or JSP file. An HREF with no leading slash (/) inherits the calling resource context. For example:
- an HREF in `http://[hostname]/myapp/servlet/MyServlet` to ″`ServletB`″ resolves to ″`http://hostname/myapp/servlet/ServletB`″
- an HREF in `http://[hostname]/myapp/servlet/MyServlet` to ″`servlet/ServletB`″ resolves to ″`http://hostname/myapp/servlet/servlet/ServletB`″ (an error)
- an HREF in `http://[hostname]/myapp/servlet/MyServlet` to ″`/ServletB`″ resolves to ″`http://hostname/ServletB`″ (an error, if `ServletB` requires the same context root as `MyServlet`)

After modifying and saving a JSP file, the change does not show up in the browser (the old JSP file displays)

It is probable that the Web application is not configured for servlet reloading, or the reload interval is too high.

To correct this problem, in an assembly tool, check the **Reloading Enabled** flag and the **Reload Interval** value in the IBM Extensions for the Web module in question. Turn Reloading on, or if it is already on, then set the Reload Interval lower.

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 JSPdirective, in this case an Include statement, was incorrect or referred to a file that could not be found.

To correct this problem, fix the problem in the JSP source, save the source and request the JSP file again.

The Java source generated from a JSP file is not retained in the temp directory (only the class file is found)

It is probable that the JSP processor is not configured to keep generated Java source.

In an assembly tool, check the **JSP Attributes** under **Assembly Property Extensions** for the Web module in question. Make sure the **keepgenerated** attribute is there and is set to true. If not, set this attribute and restart the Web application. To see the results of this operation, delete the class file from the temp directory to force the JSP processor to translate the JSP source into Java source again.

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 > DeploymentManager > Administration Services > Custom Properties**
2. Select ConnectionIOTimeout
3. Increase the ConnectionIOTimeout value
4. Click OK.

The application does not start or starts with errors

What kind of error do you see when you start an application?
- A`java.lang.ClassNotFoundException:` *`classname`* `Bean_AdderServiceHome_04f0e027Bean` error occurs
- A `ConnectionFac E J2CA0102E: Invalid EJB component: Cannot use an EJB module with version 1.1 using The Relational Resource Adapter` error occurs
- `NMSV0605E: A Reference object looked up from the context...`error when starting an application.
- Other name server (″NMSV...″) errors.

If none of these errors match the error you see:
- Browse the log files of the application server for this application looking for clues. By default, these files are: *`install_dir`*`/logs/`*`server_name`*`/SystemErr.log` and `SystemOut.log`.
- Look up any error or warning messages in the message reference table by clicking the Reference view and expanding the ″Messages″ heading.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, see ″Obtaining help from IBM″ in the information center.

java.lang.ClassNotFoundException: *classname* **Bean_AdderServiceHome_04f0e027Bean**

An similar exception occurs when you try to start an undeployed application containing enterprise beans, or containing undeployed enterprise bean modules.

Enterprise JavaBeans modules created in an assembly tool intentionally have incomplete configuration information. Deploying these modules completes the configuration by reading the module's deployment descriptor and completing platform- or installation-dependent settings and adding related classes to the Enterprise JavaBeans JAR file.

To avoid this problem, do the following:
- Use an assembly tool and administrative console to generate deployment code and install the application or Enterprise JavaBeans module onto a server.
  1. Uninstall the application or Enterprise JavaBeans module in the administrative console.
  2. Configure your assembly tool so the target server is a WebSphere Application Server installation such as **WebSphere Application Server v6.0**. See ″Configuring an assembly tool″ in the information center for additional information. If you do not have access to the target server, you can specify a false location such as `c:\temp`. Specifying a false location enables you to assemble and generate deployment code for the enterprise bean.
  3. In the Project Explorer view of an assembly tool, right-click the enterprise bean (Enterprise JavaBeans) in the undeployed .ear file containing the Enterprise JavaBeans module or the standalone undeployed Enterprise JavaBeans JAR file, and click **Deploy**. If your assembly tool can access the WebSphere Application Server target server, deployment code is generated for the Enterprise JavaBeans and the assembly tool attempts to install the application or module onto the target server. If your assembly tool cannot access the WebSphere Application Server target server or the installation fails, use the deployment code that is generated for the next step.
  4. Use the administrative console to install the deployed version created by the assembly tool.
- If you are using the `wsadmin $AdminApp install` command, uninstall it and then reinstall using the `-EJBDeploy` option. Follow the install command with the `$AdminConfig` save command.

ConnectionFac E J2CA0102E: Invalid EJB component: Cannot use an EJB module with version 1.1 using The Relational Resource Adapter

This error occurs when an enterprise bean developed to the Enterprise JavaBeans 1.1 specification is deployed with a WebSphere Application Server V5 J2C-compliant data source, which is the default data source. By default, persistent enterprise beans created under WebSphere Application Server V4.0's using the Assembly Toolkit to fulfill the Enterprise JavaBeans 1.1 specification. To run on WebSphere Application Server V5, these enterprise beans must be associated with a WebSphere Application Server V4.0-type data source.

Either modify the mapping in the application of enterprise beans to associate 1.x container managed persistence (CMP) beans to associate them with a V4.0 data source or delete the existing data source and create a V4.0 data source with the same name.

To modify the mapping in the application of enterprise beans, in the WebSphere Application Server administrative console, select the properties for the problem application and use **map resource references to resources** or **Map data sources for all 1.x CMP beans** to switch the data source the enterprise bean uses. Save the configuration and restart the application.

To delete the existing data source and create a V4.0 data source with the same name:
1. In the administrative console, click **Resources**>**Manage JDBC Providers**>*JDBC_provider_name*>**Data sources**.
2. Delete the data source associated with the Enterprise JavaBeans 1.1 module.
3. Click **Resources**>**Manage JDBC Providers**>*JDBC_provider_name*>**Data sources (Version 4)**.
4. Create the data source for the Enterprise JavaBeans 1.1 module.
5. Save the configuration and restart the application.

NMSV0605E: "A Reference object looked up from the context..." error when starting an application

If the full text of the error is similar to:

```
[7/17/02 15:20:52:093 CDT] 5ae5a5e2 UrlContextHel W NMSV0605E: A Reference object looked up from the
context"java:" with the name "comp/PM/WebSphereCMPConnectionFactory" was sent to the
JNDI Naming Manager and an exception resulted. Reference data follows:
     Reference Factory Class Name: com.ibm.ws.naming.util.IndirectJndiLookupObjectFactory
     Reference Factory Class Location URLs:
     Reference Class Name: java.lang.Object
     Type: JndiLookupInfo
     Content: JndiLookupInfo: ; jndiName="eis/jdbc/MyDatasource_CMP"; providerURL="";
              initialContextFactory=""
```

then the problem might be that the data source intended to support a CMP enterprise bean is not correctly associated with the enterprise bean.

To resolve this problem:
1. Select the **Use this Data Source in container managed persistence (CMP)** check box in the data source ″General Properties″ panel of the administrative console.
2. Verify that the JNDI Name given in administrative console under **Resources-> Manage JDBC Provider > DataSource > JNDI Name** for DataSource matches the JNDI Name given for CMP or BMP Resource Bindings at the time of Assembling the application in an assembly tool, or
3. Check the JNDI Name for CMP or BMP resource bindings specified in the code by J2EE Application Developer. Open the deployed .ear folder in an assembly tool, and look for the JNDI Name for your entity beans under CMP or BMP resource bindings. Verify that the names match.

A web resource does not display

If you are not able to display a resource in your browser, follow these steps:

1. Verify that your HTTP server is healthy by accessing the URL`http://server_name` from a browser and seeing whether the Welcome page appears. This action indicates whether the HTTP server is up and running, regardless of the state of WebSphere Application Server.
2. If the HTTP server Welcome page does not appear, that is, if you get a browser message like `page cannot be displayed` or something similar, try to diagnose your Web server problem.
3. If the HTTP server appears to function, the Application Server might not be serving the target resource. Try accessing the resource directly through the Application Server instead of through the HTTP server.

   If you cannot access the resource directly through the Application Server, Verify that the URL used to access the resource is correct.

   If the URL is incorrect and it is created as a link from another JSP file, servlet, or HTML file, try correcting it in the browser URL field and reloading, to confirm that the problem is a malformed URL. Correct the URL in the "from" HTML file, servlet or jsp file.

   If the URL appears to be correct, but you cannot access the resource directly through the Application Server, verify the health of the hosting Application Server and Web module:
   a. View the hosting Application Server and Web module in the administrative console to verify that they are up and running.
   b. Copy a simple HTML or JSP file (such as `SimpleJsp.jsp` in the WebSphere Application Server directory structure) to your Web module document root, and try to access it. If successful, the problem is with your resource. View the JVM log of your Application Server to find out why your resource cannot be found or served
4. If you can access the resource directly through the Application Server, but not through an HTTP server, the problem lies with the HTTP plug-in -- the component that communicates between the HTTP server and the WebSphere Application Server.
5. If the JSP file and the servlet output are served, but not static resources such as `.html` and image files, see the steps for enabling file serving.
6. If some kinds of resources display correctly, but you cannot display a servlet by its class name:
   - Verify that the servlet is in a directory in the Web module class path, such as in the /*Web_module_name*.war/WEB-INF/classes directory.
   - Verify that you specify the full class name of the servlet, including its package name, in the URL.
   - Verify that "`/servlet`" precedes the class name in the URL. For example, if the root context of a Web module is "`myapp`", and the servlet is `com.mycom.welcomeServlet`, then the URL reads:

     `http://hostname/myapp/servlet/com.mycom.welcomeServlet`
   - Verify that serving the servlets by class name is enabled for the hosting Web module by opening the source Web module in an assembly tool, see "Assembling applications" in the information center for additional information, and browse the *serve servlets by classname* setting in the IBM Extensions property page. If necessary, enable this flag and redeploy the Web module.
   - For servlets or other resources served by mapped URLs, the URL is http://*hostname*/*web module context root*/*mappedURL*.

If none of these steps fixes your problem, see if the problem has been identified and documented by looking at available online support (hints and tips, technotes, and fixes). If you do not find your problem listed there, see "Obtaining help from IBM" in the information center.

Diagnosing Web server problems

If you are unable to view the welcome page of your HTTP server, determine if the server is operating properly.

On Windows systems, look in the Services panel for the service corresponding to your HTTP server, and verify that the state is **Started**. If not, start it. If the service does not start, try starting it manually from the command prompt. If you are using IBM HTTP Server, the command is `IHS_install_dir\apache` .

On UNIX systems, execute the ps -ef | grep httpd command. There should be several processes running with a name of "`httpd`". If not, start your HTTP server manually. If you are using IBM HTTP Server, the command is `IHS_install_dir/bin/apachectl start`.

If the HTTP server does not start:

- Examine the HTTP server error log for clues.
- Try restoring the HTTP server to its configuration prior to installing WebSphere Application Server and restarting it. If you are using IBM HTTP Server:
  - Rename the file *IHS_install_dir*\httpd.conf.
  - Copy the httpd.conf.default file to the httpd.conf directory.
  - If Apache is running, stop and restart it.
- For the Sun ONE (iPlanet) Web server, restore the obj.conf configuration file for Sun ONE V4.1 and both obj.conf and magnus.conf files for Sun ONE V6.0 and later.
- For the Microsoft Internet Information Server (IIS), remove the WebSphere Application Server plug-in through the IIS administrative GUI.

If restoring the HTTP server default configuration file works, manually review the configuration file that has WebSphere Application Server updates to verify directory and file names for WebSphere Application Server files. If you cannot manually correct the configuration, you can uninstall and reinstall WebSphere Application Server to create a clean HTTP configuration file.

If restoring the default configuration file does not help, contact technical support for the Web server you are using. If you are using IBM HTTP Server with WebSphere Application Server, check available online support (hints and tips, technotes, and fixes). If you do not find your problem listed there, see ″Obtaining help from IBM″ in the information center

Accessing a Web resource through the application server and bypassing the HTTP server

Starting with WebSphere Application Server Version 4.0, you can bypass the HTTP server and access a web resource through the application server. It is not recommended to serve a production Web site in this way, but it provides a good diagnostic tool when it is not clear whether a problem resides in the HTTP server, WebSphere Application Server, or the HTTP plug-in.

To access a Web resource through the Application Server:

1. Determine the port of the HTTP service in the target Application Server.
   a. In the WebSphere administrative console, click **Servers>Manage Application Servers**.
   b. Select the target server, then under Additional Properties click **Web Container**.
   c. Under the Additional Properties of the Web Container, click **HTTP Transports**. You see the ports listed for virtual hosts served by the Application Server.
   d. There can be more than one port listed. In the default Application Server (server1), for example, 9060 is the port reserved for administrative requests, 9443 and 9043 are used for SSL-encrypted requests. To test the sample ″`snoop`″ servlet, for example, use the default application port 9080, unless it changes.
2. Use the HTTP transport port number of the Application Server to access the resource from a browser. For example, if the port is 9080, the URL is `http://`*`hostname`*`:9080/myAppContext/myJSP.jsp`.
3. If you are still unable to access the resource, verify that the HTTP transport port is in the ″Host Alias″ list:
   a. Click **Application Servers > Your_ApplicationServer > Web Container > HTTP Transports** to check the Default virtual host and the HTTP transport ports used by this Application Server.
   b. Click **Environment > Manage Virtual Hosts > default host > Host Aliases** to check if the HTTP transport port exists. Add an entry if necessary. For example, if the HTTP port for your application is server is 9080, add a host alias of *:9082.

Cannot uninstall an application or remove a node or application server

What kind of problem are you having?

- After uninstalling an application through wsadmin tool, the application continues to run and throws ″DocumentIOException″
- 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 10. 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 cannot uninstall or remove a node or application server.
  - I have problems creating or using HTTP sessions.
  - I have problems using tracing, logging, log files, or other troubleshooting features.
  - I get errors connecting to the administrative console from a Netscape browser.
  - The stopServer.sh (base and ND) hangs and creates a Java core dump (Red Hat Linux).
  - A and cannot be recovered.
- If you did not solve the problem, prepare to contact IBM support.

  If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

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

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

Administration and administrative console troubleshooting 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. 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*.**.
  - *install_root*/profiles/*profile_name*/logs/*server_name*/SystemOut.log 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*.**.
  - *install_root*/profiles/*profile_name*/logs/dmgr/SystemOut.log for the message that indicates that the server started successfully: **ADMU3000I: Server dmgr open for e-business; process id is nnnn**.
- 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 SystemOut.log file.
  - The line **ADMC0013I: SOAP connector available at port *nnnn*** indicates the port that the server is using to listen for wsadmin functions.
  - The 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 SystemOut.log file , 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 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 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
- ″Tracing the administrative component: WebSphere Application Server technical support might ask you to trace the administrative component for detailed problem determination. The trace specification for this component is com.ibm.websphere.management.*=all=enabled:com.ibm.ws.management.*=all=enabled″. See ″Tracing the administrative component″ in the information center.

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.

If none of these steps solves the problem, see if the problem has been identified and documented by using the links in ″Diagnosing and fixing problems: Resources for learning″ in the information center. 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″ in the information center.

An "Internal Server Error", "Page cannot be found", 404, or similar error occurs trying to view the administrative console

Here are some steps to try if you are unable to view the administrative console:
- Verify that the application server that supports the administrative console is up and running. For a base configuration, the administrative console is deployed by default on server1. Before viewing the administrative console, you must take one of the following actions:
  - Run the **startServer server1** command for the Windows platform or **./startServer.sh server1** command for a UNIX platform from a command prompt in the *install_dir*\bindirectory
  - Click the **Start application server** link from the first steps panel
  - Start WebSphere Application Server as a service or from the Start menu, if you are using a Windows operating system.
- 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 SystemOut.log file for the application server or the deployment manager to verify that the server supporting the administrative console has started. See ″Viewing the JVM logs″ in the information center.
- Check the URL you use to view the console. By default, it is http://*server_name*:9060/ibm/console, where *server_name* is the host name.
- If you are browsing the console from a remote machine, try to eliminate connection, address and firewall issues by pinging the server machine from a command prompt, using the server name in the URL.
- If you have never been able to access the administrative console see ″Troubleshooting installation″ in the information center..

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.

- If you are not the administrator, ask the administrator to enable your ID for the administrative console.

The directory paths in the console contain strange characters

Directory paths used for class paths or resources specified in an assembly tool, configuration files, or elsewhere that contain strange characters when they are viewed in the administrative console might result from the Java run time interpreting a backslash (\) as a control character.

To resolve, modify Windows-style class paths by replacing occurrences of single back slashes to two. For example, change c:\MyFiles\MyJsp.jsp to c:\\MyFiles\\MyJsp.jsp.

Errors connecting to the administrative console from a browser

This topic describes problems that you can have when logging into the administrative console from a browser.

Review the following information to resolve your browser problem.

If you are able to bring up the browser page, but the console behavior is inconsistent, error-prone, or unresponsive, try upgrading the browser you are using. Older browsers may not support the administrative console's features. For a listing of supported Web browsers, see http://www.ibm.com/software/webservers/appserv/doc/latest/prereq.html.

Check to see if the problem has been identified and documented using the links in ″Diagnosing and fixing problems: Resources for learning″ in the information center. If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

Check the following list for your problem and how to solve it:

- When a single user that uses multiple instances of the Mozilla browser logs into the administrative console, the first user ID that logs into the administrative console is the current user.
- A user on Mozilla browser Version 1.4 selects a check box on a collection table, presses Enter, and receives an error.
- A user on Mozilla browser Version 1.4 enters an invalid ID or password, presses Enter, and receives an error message

When a single user that uses multiple instances of the Mozilla browser logs into the administrative console, the first user ID that logs into the administrative console is the current user.

When a single user logged into an operating system tries to use multiple instances of the Mozilla browser, the first user ID that logs into the administrative console is the current user. This situation occurs because the browser windows share a single process.

To resolve the problem, do one of the following actions:

- Have single users logged into an operating system use a single instance of the Mozilla browser to log into the administrative console.
- If the operating system allows multiple users on an operating system, have each user log into the operating system with a different user ID and bring up a single instance of the Mozilla browser.

A user on Mozilla browser Version 1.4 selects a check box on a collection table, presses Enter, and receives an error.

A user on Mozilla browser Version 1.4 selects a check box on a collection table, presses Enter, and receives an error.

To resolve the problem, do one of the following actions:

- Explicitly select a button of interest on the administrative console panel instead of pressing Enter.
- Use a later version of a supported Mozilla browser.
- Use a supported version of the Microsoft Internet Explorer browser.

A user on Mozilla browser Version 1.4 enters an invalid ID or password, presses Enter, and receives an error message

A user on Mozilla browser Version 1.4 enters an invalid user ID or password, presses Enter, and receives an error message. Clicking OK fails to refresh the administrative login screen

To resolve the problem, do one of the following actions:

- Use a later version of a supported Mozilla browser.
- Use a supported version of the Microsoft Internet Explorer browser.

Web server plug-in troubleshooting tips

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 the`http_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 the`http_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.

- 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 server is started. View the administrative console for error messages or look in the JVM logs.
- In the administrative console, select the problem application server and view its installed applications to verify that they are started.

If none of these steps solves the problem:
- For specific problems that can cause web pages and their contents not to display, seeWeb resource (JSP file, servlet, html file, image, etc) will not display in the information center.
- Check to see if the problem has been identified and documented using the links in ″Diagnosing and fixing problems: Resources for learning″ in the information center.
- If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

For current information available from IBM Support on known problems and their resolution, see the following topics on the IBM support page:
- HTTP transport
- HTTP plug-in
- HTTP plug-in remote install

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

The server process does not start or starts with errors

If the WebSphere Application Server installation program completes successfully, but the Application Server does not start, or starts with errors

- Browse the Application Server log files for clues. See ″Viewing the JVM logs″ in the information center. The log files are located by default in:
  - Linux UNIX *install_root*\ profiles\ *profile_name*\ logs\ *server_name*\ SystemErr.log and SystemOut.log
  - Windows *install_root*/ profiles/ *profile_name*/ logs/ *server_name*/ SystemErr.log and SystemOut.log

  Several applications deployed on an Application Server or node can take time to start. Browse the SystemOut.log periodically and look at the most recent updates to see if the server is still starting up.
- Linux UNIX The tail -f *install_root*/ profiles/ *profile_name*/ logs/ *server_name*/ SystemOut.log is a convenient way to watch the progress of the server.
- Look for any errors or warnings relating to specific resources with the module, such as Web modules, enterprise beans and messaging resources. If you find any, examine the Application Server configuration file for the configuration settings of that resource. For example, in a base or non-distributed configuration on Windows systems, browse *install_root*/ profiles/ *profile_name*/ config/ cells/ *ApplicationServerCell*/ nodes/ *node_name*/ servers/ *server_name*/server.xml, and examine the XML tags for the properties of that resource. Change its **initialState** value from START to STOP. Then restart the server to see if this component causes the problem.
- Look up any error or warning messages in the message reference table by clicking the **Reference** view of the information center navigation and expanding **Messages** in the navigation tree.
- Verify that the logical name that you specified to appear on the console for your Application Server does not contain invalid characters like: - / \ : * ? ″ < > and leading or trailing spaces.
- If you are using IBM Cloudscape and receive an `ERROR XSDB6: Another instance of Cloudscape may have already booted the database` *`databaseName`* error when starting the Application Server, see ″Cannot access a data source″ in the information center for additional information.
- When using a non-root user ID to run Application Servers, verify that:
  - The non-root user has write access to the *install_root*/temp directory.
  - The JVM has write access to *install_root*/config/plugin-cfg.xml file.
  - The non-root user has access to the `logs` directory.

Message ″The socket bind failed for host *hostname* **and port** *portnumber***. The port may already be in use.″ occurs when restarting an Application Server.**

The following error message might appear in the SystemOut.log after restarting the Application Server:

```
The socket bind failed for host hostname and port portnumber.  The port may already be in use.
```

This problem might occur if the network is slow, and the port listed in the message text did not finish listening when the application was stopped and restarted.

To verify that this is the problem, check the port status.

To correct this problem, wait for a few minutes after stopping the server:

1. Verify that no ports are listening. Use the command: ’netstat -a.
2. Restart the server

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″ in the information center..

The stopServer.sh hangs and creates a Java core dump (Red Hat Linux)

When you run the stopServer.sh script on Red Hat Linux Advanced Server Version 2.1 with the latest operating system patches, it creates a Java core dump and hangs the terminal. To fix this problem:

- kill all Java and MQ processes
- uninstall the latest version of GNU Standard C++ Library
  - run the command `rpm -e --nodeps libstdc++-2.96-116.7.2`
- Reinstall the older version from Redhat Advanced Server V2.1 CD
  - run the command `rpm -ihv libstdc++-2.96-108.1.rpm`

Problems starting or using the wsadmin command

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.
- ″The input line is too long″ error returned from the `wsadmin` command on a Windows platform.
- WASX701E: Exception received while running file ″*scriptName*.jacl″; exception information: com.ibm.bsf.BSFException: error while evaluating Jacl expression: missing close-bracket

If you do not see your problem here:

- If you are not able to enter wsadmin command mode, try running **wsadmin -c** ″**$Help wsadmin**″ for help in verifying that you are entering the command correctly.
- If you can get the wsadmin command prompt, enter **$Help help** to verify that you are using specific commands correctly.
- wsadmin commands are a superset of Jacl (Java Command Language), which is in turn a Java-based implementation of the Tcl command language. For details on Jacl syntax beyond wsadmin commands, refere to the Tcl developers' site, http://www.tcl.tk. For specific details relating to the Java implementation of Tcl, refer to http://www.tcl.tk/software/java.
- Browse the *install_dir*/profiles/*profile_name*/logs/wsadmin.traceout file for clues.
  - Keep in mind that wsadmin.traceout is refreshed (existing log records are deleted) whenever a new wsadmin session is started.
  - If the error returned by wsadmin does not seem to apply to the command you entered, for example, you receive WASX7023E, stating that a connection could not be created to host ″myhost,″ but you did not specify ″-host myhost″ on the command line, examine the properties files used by wsadmin to determine what properties are specified. If you do not know what properties files were loaded, look for the WASX7326I messages in the wsadmin.traceout file; there will be one of these messages for each properties file loaded.

If none of these steps fixes your problem, check to see if the problem has been identified and documented by looking at the available online support (hints and tips, technotes, and fixes). If you don't find your problem listed there contact IBM support.

"WASX7023E: Error creating "SOAP" connection to host" or similar error trying to launch wsadmin command line utility

By default, the wsadmin utility attempts to connect to an application server at startup. This is because some commands act upon running application servers. This error indicates that no connection could be established.

To resolve this problem:

- If you are not sure whether an application server is running, start it by entering **startserver** *servername* from the command prompt. If the server is already running, you will see an error similar to ″ADMU3027E: An instance of the server is already running″.
- If you are running a Network Deployment configuration, you will first need to start the deployment manager by running ″startManager″ or ″startManager.sh″ from the *install_dir*/bin directory. Then you can launch wsadmin immediately to connect to the deployment manager, or start a node and application server to connect to.
- If an application server is running and you still get this error:
  - If you are running remotely (that is, on a different machine from the one running WebSphere Application Server), you must use the **-host** *hostname* option to the wsadmin command to direct wsadmin to the right physical server.
  - If you are using the -host option, try pinging the server machine from the command line from the machine on which you are trying to launch wsadmin to verify there are no issues of connectivity such as firewalls.
  - verify that you are using the right port number to connect to the WebSphere Application Server process:
    - If you are not specifying a port number (using the -port option) when you start the wsadmin tool, the wsadmin tool uses the default port specified in *install_dir*/profiles/*profile_name*/properties/wsadmin.properties file, property name=com.ibm.ws.scripting.port (default value =8879).
    - The port that wsadmin should send on depends on the server process wsadmin is trying to connect to.

      For a single-server installation, wsadmin attempts to connect to the application server process by default. To verify the port number:
      - Look in the file *install_dir*/profiles/*profile_name*/config/cells/*node_name*/nodes/*node_name*/serverindex.html for a tag containing the property **serverType=″APPLICATION_SERVER″**.
      - Look for an entry within that tag with the property **endPointName=″SOAP_CONNECTOR_ADDRESS″**.
      - Look for a **port** property within that tag. This is the port wsadmin should send on.

      In a Network Deployment installation, wsadmin launched from the bin directory on the Network Deployment installation attempts to send requests to the deployment manager by default. To verify the port number:
      - Get the hostname of the node on which the Deployment Manager is installed.
      - Using that hostname, look in *install_dir*/profiles/*profile_name*/config/cells/*cell_name*/nodes/*node_name*/serverindex.html file for a tag containing the property **serverType=″DEPLOYMENT_MANAGER″**.
      - Within that tag, look for an entry with a property **endPointName=″SOAP_CONNECTOR_ADDRESS″**.
      - Within that tag, look for a ″port″ property. This is the port that the wsadmin tool should send on.

"com.ibm.bsf.BSFException: error while eval'ing Jacl expression: no such method *command name* in class com.ibm.ws.scripting.AdminConfigClient" returned from wsadmin command.

This error is usually caused by a misspelled command name. Use the **$AdminConfig help** command to get information about what commands are available. Note that command names are case-sensitive.

WASX7022E returned from running "wsadmin -c ..." command, indicating invalid command

If the command following -c appears to be valid, the problem may be caused by the fact that on Unix, using wsadmin -c to invoke a command that includes dollar signs results in the shell attempting to do variable substitution. To confirm that this is the problem, check the command to see if it contains an unescaped dollar sign, for example: **wsadmin -c ″$AdminApp install ....″**.

To correct this problem, escape the dollar sign with a backslash. For example: **wsadmin -c ″\$AdminApp install ...″**.

com.ibm.ws.scripting.ScriptingException: WASX7025E: String "" is malformed; cannot create ObjectName

One possible cause of this error is that an empty string was specified for an object name. This can happen if you use one scripting statement to create an object name and the next statement to use that name, perhaps in an ″invoke″ or ″getAttribute″ command, but you don't check to see if the first statement really returned an object name. For example (the following samples use basic Jacl commands in addition to the wsadmin Jacl extensions to make a sample script):

```
#let's misspell "Server"
set serverName [$AdminControl queryNames type=Srever,*]
$AdminControl getAttributes $serverName
```

To correct this error, make sure that object name strings have values before using them. For example:

```
set serverName[$AdminControl queryNames node=mynode,type=Server,name=server1,*]
if {$serverName == ""} {puts "queryNames returned empty - check query argument"}
else {$AdminControl getAttributes $serverName}
```

For details on Jacl syntax beyond wsadmin commands, refer to the Tcl developers' site, http://www.tcl.tk.

"The input line is too long" error returned from the wsadmin command on a Windows platform

This error indicates that the Windows command line limit of 2048 characters has been exceeded, probably due to a long profile path used within the **wsadmin.bat** command. You may get this error when running wsadmin in a Windows command prompt or calling wsadmin from a `.bat` file, an `ant` build file, or Profile Creation Tool. If this error results in running wsadmin in a way other than from the Profile Creation Tool, you can avoid the problem by using the Windows **subst** command which allows you to map an entire path to a virtual drive. To see the syntax of the **subst** command, enter `help subst` from a Windows command prompt.

For example if the product resides in the `c:\Program Files\IBM\WebSphere\AppServer` directory, edit the `c:\Program Files\IBM\WebSphere\AppServer\bin\setupCmdLine.bat` file. For example:

```
SET CUR_DIR=%cd%
cd /d "%~dp0.."
SET WAS_HOME=%cd%
cd /d "%CUR_DIR%"

@REM add the following two lines to workaround Windows 2K command line length limit
subst w: %WAS_HOME%
set WAS_HOME=w:

...
...
```

Then edit the `setupCmdLine.bat` file which resides in the `bin` directory of your profile. For example:

```
SET WAS_USER_PROFILE=...
SET USER_INSTALL_ROOT=...
SET WAS_HOME=c:\Program Files\IBM\WebSphere\AppServer
SET JAVA_HOME=c:\Program Files\IBM\WebSphere\AppServer\java
```

```
@REM add the following three lines to workaround Windows 2K command line length limit
subst w: %WAS_HOME%
set WAS_HOME=w:
set JAVA_HOME=%WAS_HOME%\java

...
...
```

If this error occurred while running the Profile Creation Tool, you have to rerun the Profile Creation Tool to provide a shorter profile path with a shorter profile name. If this does not fix the problem, then you have to follow the same instruction given above to edit the setupCmdLine.bat file resided in the bin directory of your WebSphere install. After editing the file, rerun Profile Creation Tool. If the same problem persists, then you may have to reinstall WebSphere with a shorter install path.

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.
-

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 using tracing, logging or other troubleshooting features

What kind of problem are you having?
- Error messages when launching the Log Analyzer
- Netscape browser fails when trying to enable a component trace

Error messages when launching the Log Analyzer

Upon starting the Log Analyzer for the first time or after the Log Analyzer preferences files of the users are deleted, the following message displays in the Log Analyzer shell window:

```
Cannot open input stream for waslogbrsys
```

This message is an informational message. You can disregard the message because it does not affect the execution of the Log Analyzer.

The following error messages might display in the Log Analyzer shell window when you start the Log Analyzer:

```
Cannot open input stream for default
Cannot open input stream for default
Cannot load configuration: default
Cannot open input stream for default
Cannot open input stream for default
Cannot load configuration: default
```

These error messages indicate corrupt or incomplete user preference files.

To resolve this problem, take the following steps:

1. Close the Log Analyzer.
2. Delete all user preference files in the *%USERPROFILE%\logbr* directory on Windows platforms or *$HOME/logbr* directory on UNIX platforms.
3. Restart the Log Analyzer.

Note: Deleting all user preference files removes the preferences of Log Analyzer set by the user in the preferences dialog.

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.

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 Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, New York 10594 USA

Trademarks and service marks

Tor trademark attribution, visit the IBM Terms of Use Web site (http://www.ibm.com/legal/us/).