BEA WebLogic Server™ and WebLogic Express®

Administration Guide

Release 7.0
Document Revised: April 27, 2006

Administration Guide

Part Number	Date	Software Version
N/A	June 28, 2002	BEA WebLogic Server Version 7.0

Contents

2. Starting and Stopping WebLogic Servers

B. WebLogic Server Command-Line Interface Reference

About This Document

This document explains the management subsystem provided for configuring and monitoring your WebLogic Server implementation. It is organized as follows:

- Chapter 1, "Overview of WebLogic System Administration," describes the architecture of the WebLogic Server management subsystem.
- Chapter 2, "Starting and Stopping WebLogic Servers," explains the procedures for starting and stopping WebLogic Servers.
- Chapter 3, "Protecting System Administration Operations," explains how to limit access to system administration tasks using roles.
- Chapter 4, "Using Log Messages to Manage WebLogic Server," describes the use of the WebLogic Server local log and the domain-wide log for managing a WebLogic Server domain.
- Chapter 5, "Deploying Applications," describes installation of applications on the WebLogic Server and the deploying of application components.
- Chapter 6, "Configuring WebLogic Server Web Components," explains the use of WebLogic Server as a Web Server.
- Chapter 7, "Managing Transactions," explains how to manage the Java Transaction subsystem within a WebLogic Server domain.
- Chapter 8, "Managing JDBC Connectivity," discusses the management of Java Database Connectivity (JDBC) resources within a WebLogic Server domain.
- Chapter 9, "Managing JMS," discusses the management of Java Message Service within a WebLogic Server domain.
- Chapter 10, "Using the WebLogic Messaging Bridge," discusses how to configure a store and forward mechanism between any two JMS providers.

- Chapter 11, "Managing JNDI," discusses how to use the WebLogic JNDI naming tree, including viewing and editing objects on the JNDI naming tree and binding objects to the JNDI tree.
- Chapter 12, "Managing the WebLogic J2EE Connector Architecture," describes how extensions to the WebLogic J2EE platform that allow connections to other Enterprise Information Systems are managed.
- Chapter 13, "Managing WebLogic Server Licenses," describes how to update your BEA license.
- Appendix A, "Using the WebLogic Java Utilities," describes a number of utilities that are provided for developers and system administrators.
- Appendix B, "WebLogic Server Command-Line Interface Reference," describes the syntax and usage of the command-line interface for managing a WebLogic Server domain.
- Appendix C, "Using Ant Tasks to Configure a WebLogic Server Domain," describes the syntax and usage of WebLogic Server Ant tasks that help you manag a WebLogic Server domain.
- Appendix D, "WebLogic SNMP Agent Command-Line Reference," describes using the WebLogic SNMP agent's command line interface to retrieve the value of WebLogic Server attributes and generate and receive WebLogic Server traps.

Audience

This document is intended mainly for system administrators who will be managing the WebLogic Server application platform and its various subsystems.

e-docs Web Site

BEA product documentation is available on the BEA corporate Web site. From the BEA Home page, click on Product Documentation.

How to Print the Document

You can print a copy of this document from a Web browser, one main topic at a time, by using the File→Print option on your Web browser.

A PDF version of this document is available on the WebLogic Server documentation Home page on the e-docs Web site (and also on the documentation CD). You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the WebLogic Server documentation Home page, click Download Documentation, and select the document you want to print.

Adobe Acrobat Reader is available at no charge from the Adobe Web site at http://www.adobe.com.

Contact Us!

Your feedback on BEA documentation is important to us. Send us e-mail at docsupport@bea.com if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the documentation.

In your e-mail message, please indicate the software name and version you are using, as well as the title and document date of your documentation. If you have any questions about this version of BEA WebLogic Server, or if you have problems installing and running BEA WebLogic Server, contact BEA Customer Support through BEA WebSupport at http://www.bea.com. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using

- A description of the problem and the content of pertinent error messages

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Usage
Ctrl+Tab	Keys you press simultaneously.
italics	Emphasis and book titles.

`monospace text`	Code samples, commands and their options, Java classes, data types, directories, and file names and their extensions. Monospace text also indicates text that you enter from the keyboard.
	Examples:
	`import java.util.Enumeration;`
	`chmod u+w *`
	`config/examples/applications`
	`.java`
	`config.xml`
	`float`

`monospace italic text`	Variables in code. *Example*: `String` *`CustomerName`*`;`
UPPERCASE TEXT	Device names, environment variables, and logical operators. *Examples*: LPT1 BEA_HOME OR
`{ }`	A set of choices in a syntax line.

Convention	Usage

`[ ]`	Optional items in a syntax line. *Example*:
	
	`java utils.MulticastTest -n` *`name`* `-a` *`address`*
	`     [-p` *`portnumber`*`] [-t` *`timeout`*`] [-s` *`send`*`]`

`|`	Separates mutually exclusive choices in a syntax line. *Example*:
	
	`java weblogic.deploy [list|deploy|undeploy|update]`
	`     password {application} {source}`

`...`	Indicates one of the following in a command line: ■ An argument can be repeated several times in the command line. ■ The statement omits additional optional arguments. ■ You can enter additional parameters, values, or other information
. . .	Indicates the omission of items from a code example or from a syntax line.

CHAPTER

1 Overview of WebLogic System Administration

The following sections provide an overview of system administration for WebLogic Server:

Introduction to System Administration

WebLogic Server system administration tools allow you to install, configure, monitor, and manage one or more WebLogic Server installations. You also use the tools to manage and monitor the applications hosted on WebLogic Server. A WebLogic Server installation can consist of a single WebLogic Server instance or multiple instances, each hosted on one or more physical machines.

Using the system administration tools, which include an Administration Console, command line utilities, and an API, you manage services such as security, database connections, messaging, and transaction processing. The tools also include capabilities for monitoring the health of the WebLogic Server environment to ensure maximum availability for your applications.

WebLogic Server Domains

The basic administrative unit for WebLogic Servers is called a *domain*. A domain is a logically related group of WebLogic Server resources that are managed as a unit by a WebLogic Server instance configured as the *Administration Server*. A domain includes one or more WebLogic Servers and may also include WebLogic Server *clusters*. Clusters are groups of WebLogic Servers that work together to provide scalability and high-availability for applications. Applications are also deployed and managed as part of a domain.

You can organize your domains based on criteria such as:

- *Logical divisions of applications*. For example, a domain devoted to end-user functions such as shopping carts and another domain devoted to back-end accounting applications.
- *Physical location*. Domains for different locations or branches of your business.
- *Size*. Domains into organized in small units that can be managed more efficiently, perhaps by different system administration personnel.

Note: All WebLogic Server instances in a domain must run the same version of the WebLogic Server software. The Administration Server must also have the same or later service pack installed as the Managed Servers in its domain. For example, the Administration Server could be running version 7.0, Service Pack 1 while the Managed Servers are running version 7.0 without Service Pack 1.

For more information about domains, see Creating and Configuring WebLogic Server Domains at `http://e-docs.bea.com/wls/docs70/admin_domain/index.html`.

Figure 1-1 WebLogic Server Domain

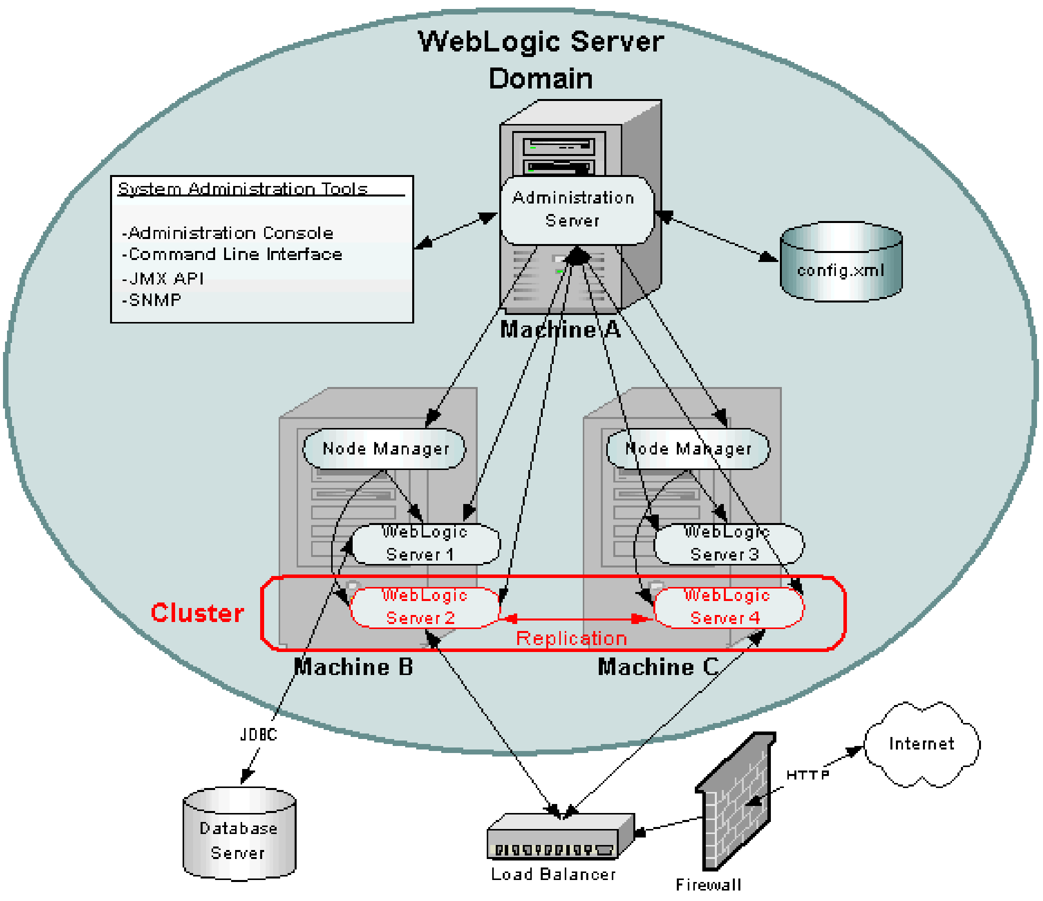


Figure 1-1 depicts a possible configuration of a WebLogic Server Domain—one of many possible configurations.

In the depicted domain, there are three physical machines.

Machine A is dedicated as the Administration Server and hosts one instance of WebLogic Server. The System Administration Tools communicate with the Administration Server to perform configuration and monitoring of the servers and applications in the domain. The Administration Server communicates with each of the Managed Servers on behalf of the System Administration Tools. The configuration for all the servers in the domain is stored in the configuration repository, the `config.xml` file, which resides on the machine hosting the Administration Server.

Machines B and C each host two instances of WebLogic Server, WebLogic Servers 1 through 4. These instances are called *Managed Servers*. The Administration Server communicates with an instance of Node Manager running on each machine to control startup and shutdown of the Managed Servers.

WebLogic Servers 2 and 4 are part of a WebLogic Cluster (outlined in red). This cluster is running an application that responds to HTTP requests routed to the cluster from a hardware load balancer. (You could also use an instance of WebLogic Server to provide load balancing.) The load balancer processes HTTP requests from the Internet after they have passed through a firewall. The load balancer and firewall are not part of the domain. A replicated copy of objects such as HTTP sessions is passed between the two cluster members to provide failover capability.

WebLogic Server 1 runs an application that uses JDBC to access a database server running on another physical machine that is not part of the WebLogic Domain.

Note: The pictured domain is only intended to illustrate the concepts of a WebLogic Server domain and how you manage the domain. Many possible configurations of servers, clusters, and applications are possible in a WebLogic Server domain.

System Administration Infrastructure

System administration infrastructure in WebLogic Server is implemented using the Java Management Extension (JMX) specification from Sun Microsystems. The JMX API models system administration functions with Java objects called MBeans. Knowledge of this implementation as described in this discussion of system administration infrastructure is *not* necessary to manage a WebLogic Server domain.

There are three types of MBeans used to manage a WebLogic Server domain: *administration*, *configuration,* and *runtime* Mbeans.

Administration Mbeans contain a set of *attributes* that define configuration parameters for various management functions. All attributes for administration MBeans have pre-set default values. When the Administration Server starts, it reads a file called `config.xml` and overrides the default attribute values of the administration MBeans with any attribute values found in the `config.xml` file.

The `config.xml` file, located on the machine that hosts the Administration Server, provides persistent storage of Mbean attribute values. Every time you change an attribute using the system administration tools, its value is stored in the appropriate administration MBean and written to the `config.xml` file. Each WebLogic Server domain has its own `config.xml` file.

If you set any configuration attributes on the command line when you start the Administration Server using the `-D` arguments, these values override the values set by the defaults or those read from the `config.xml` file. These overridden values are also persisted to `config.xml` file by the Administration Server. For more information about these command-line arguments, see "Using the weblogic.Server Command" on page 2-17.

Configuration Mbeans are copies of Administration Mbeans that each Managed Server uses to initialize its configuration. When you start a Managed Server, the server receives a copy of the of all the administration MBeans from the Administration Server and stores them in memory as configuration MBeans. If you override any configuration attributes when starting a Managed Server, those values override the values received from the Administration Server but are not written to the `config.xml` file. For more information about starting a Managed Server, see "Starting a Managed Server" on page 2-33.

Runtime Mbeans contain sets of attributes consisting of runtime information for active WebLogic Servers instances and applications. By retrieving the values of attributes in these runtime MBeans, you can monitor the running status of a WebLogic Server domain.

Mbeans may also contain *operations* used to execute management functions.

Although users with a knowledge of these Mbeans and the JMX API can create their own customized management system, most users prefer to use the system administration tools provided with WebLogic Server to perform these tasks. These tools do not require knowledge of the JMX API. For more information, see "System Administration Tools" on page 1-8.

The Administration Server and Managed Servers

One instance of WebLogic Server in each domain is configured as an *Administration Server*. The Administration Server provides a central point for managing a WebLogic Server domain. All other WebLogic Server instances in a domain are called *Managed Servers*. In a domain with only a single WebLogic Server instance, that server functions both as Administration Server and Managed Server.

For a typical production system, BEA recommends that you deploy your applications only on Managed Servers. This practice allows you to dedicate the Administration Server to configuration and monitoring of the domain.

For more information, see "Starting and Stopping WebLogic Servers" on page 2-1.

Failover for the Administration Server

To prevent the Administration Server from becoming a single point of failure, Managed Servers can always function without the presence an Administration Server, but an Administration Server is required to manage and monitor the domain. By maintaining backups of the `config.xml` file and certain other resources for a domain, you can replace a failed Administration Server with a backup WebLogic Server instance that can assume the role of Administration Server. For more information, see "Starting an Administration Server" on page 2-12 and Recovering Failed Servers at `http://e-docs.bea.com/wls/docs70/admin_domain/failures.html`.

Failover for Managed Servers

When a Managed Server starts, it contacts the Administration Server to retrieve its configuration information. If a Managed Server is unable to connect to the specified Administration Server during startup, it can retrieve its configuration directly by reading a configuration file and other files located on the Managed Server's file system.

A Managed Server that starts in this way is running in *Managed Server Independence* mode. In this mode, a server uses cached application files to deploy the applications that are targeted to the server. You cannot change a Managed Server's configuration until it is able to restore communication with the Administration Server. For more information, see Recovering Failed Servers at `http://e-docs.bea.com/wls/docs70/admin_domain/failures.html`.

Domain-Wide Administration Port

You can enable an administration port for use with servers in a domain. The administration port is optional, but it provides two important capabilities:

- It enables you to start a server in standby state. While in the standby state, the administration port remains available to activate or administer the server. However, the server's other network connections are unavailable to accept client connections. See Starting and Stopping WebLogic Server for more information on the standby state.
- It enables you to separate administration traffic from application traffic in your domain. In production environments, separating the two forms of traffic ensures that critical administration operations (starting and stopping servers, changing a server's configuration, and deploying applications) do not compete with high-volume application traffic on the same network connection.

  For more information, see Configuring a Domain-Wide Administration Port in *Creating and Configuring WebLogic Server Domains* at `http://e-docs.bea.com/wls/docs70/admin_domain/network.html#administration_port_and_administration_channel.`

Service Packs and WebLogic Server Instances

All WebLogic Server instances in a domain must run the same version of the WebLogic Server software. The Administration Server must also have the same or later service pack installed as the Managed Servers in its domain. For example, the Administration Server could be running version 7.0, Service Pack 1 while the Managed Servers are running version 7.0 without Service Pack 1.

System Administration Tools

Using JMX as the underlying architecture, system administration tools are provided for a variety of management functions. An Administration Server must be running when you use system administration tools to manage a domain.These tools are discussed in the next sections.

Security Protections for System Administration Tools

All system administration operations are protected based on the user name used to access a system administration tool. A user (or the group a user belongs to) must be a member of one of four security roles. These roles grant or deny a user access to various sets of system administration operations. The roles are Admin, Operator, Deployer, and Monitor. You can also set a security policy on WebLogic Server instances in a domain. For more information, see "Protecting System Administration Operations" on page 3-1.

System Administration Console

The Administration Console is a Java ServerPages (JSP)-based tool hosted by the Administration Server. You can access the Administration Console using a Web browser from any machine on the local network that can communicate with the Administration Server (including a browser running on the same machine as the Administration Server). The Administration Console allows you to manage a WebLogic Server domain containing multiple WebLogic Server instances, clusters, and applications. The management capabilities include:

- Configuration
- Stopping and starting servers
- Monitoring server health and performance
- Monitoring application performance
- Viewing server logs
- Editing deployment descriptors for Web Applications, EJBs, J2EE Connectors, and Enterprise Applications.

Using the Administration Console, system administrators can easily perform all WebLogic Server management tasks without having to learn the JMX API or the underlying management architecture. The Administration Server persists changes to attributes in the `config.xml` file for the domain you are managing.

For more information, see:

- "Starting and Using the Administration Console" on page 1-22
- Administration Console Online Help at `http://e-docs.bea.com/wls/docs70/ConsoleHelp/index.html`. (The online help is also available from the Administration Console by clicking on the "?" icons.)

Command-Line Interface

The command-line interface allows you to manage a WebLogic Servers domain when using the Administration Console is not practical or desired—such as when you want to use scripts to manage your domain, when you cannot use a Web browser to access the Administration Console, or if you prefer using the command-line interface over a graphical user interface. You can use only the command-line interface to manage your domain, or you may use the command-line interface in combination with other system administration tools such as the Administration Console to manage you domain.

The command line interface invokes a Java class called `weblogic.Admin`. Arguments for this class provide the ability to perform many common management functions without the need to learn the JMX API. For more information, see:

- "WebLogic Server Administration Command Reference" on page B-7
- WebLogic Server API Reference (Javadocs) at `http://e-docs.bea.com/wls/docs70/javadocs/index.html`. (See the `weblogic.management` packages.)

If you require more fine-grained control than the `weblogic.Admin` management functions provide you can also use the command line interface to perform *`set`* or *`get`* operations directly on Mbean attributes. This feature requires knowledge of the WebLogic Server Mbean architecture. For more information, see the following resources:

- "MBean Management Command Reference" on page B-49 provides a guide to using the command-line interface
- Javadocs for WebLogic Server Classes at `http://e-docs.bea.com/wls/docs70/javadocs/index.html`.
  - Select the `weblogic.management.configuration` package for configuration MBeans (to configure a WebLogic Domain)
  - Select the `weblogic.management.runtime` package for runtime MBeans (for monitoring).
- A reference of Mbeans and attributes is provided in the BEA WebLogic Server Configuration Reference at `http://e-docs.bea.com/wls/docs70/config_xml/index.html`. This reference is correlated with the elements representing MBeans in the `config.xml` file.

JMX

Advanced Java programmers with knowledge of the JMX API from Sun Microsystems Inc. and WebLogic Server Mbeans can write their own management components as a Java class.

For more information, see:

- Programming WebLogic JMX Services at `http://e-docs.bea.com/wls/docs70/jmx/index.html`.
- WebLogic Server API Reference (Javadocs) at `http://e-docs.bea.com/wls/docs70/javadocs/index.html`. (See the `weblogic.management` packages.)

Configuration Wizard

Use the Configuration Wizard to create a new WebLogic Server domain. This tool can create domain configurations for stand-alone servers, Administration Servers with Managed Servers, and clustered servers. The Configuration Wizard creates the appropriate directory structure for your domain, a basic `config.xml` file, and scripts you can use to start the servers in your domain.

You can run the Configuration Wizard either using a graphical user interface (GUI) or in a text-based command line environment. You can also create user-defined domain configuration templates for use by the Configuration Wizard.

For more information, see Creating New Domains Using the Configuration Wizard in *Creating and Configuring WebLogic Server Domains* at `http://e-docs.bea.com/wls/docs70/admin_domain/configwiz.html`.

Java Utilities

Utility programs are provided for common tasks such as deploying an application and testing DBMS configurations. For more information, see "Using the WebLogic Java Utilities" on page A-1.

Node Manager

Node Manager is a Java program provided with WebLogic Server that enables you to start, shut down, restart, and monitor remote WebLogic Server instances. To enable these capabilities, you run an instance of Node Manager on each physical machine in your domain.

For more information, see Managing Server Availability with Node Manager at `http://e-docs.bea.com/wls/docs70/admin_domain/nodemgr.html`.

SNMP

WebLogic Server includes the ability to communicate with enterprise-wide management systems using Simple Network Management Protocol (SNMP). The WebLogic Server SNMP capability enables you to integrate management of WebLogic Servers into an SNMP-compliant management system that gives you a single view of the various software and hardware resources of a complex, distributed system.

For more information, see:

- WebLogic SNMP Management Guide at `http://e-docs.bea.com/wls/docs70/snmpman/index.html`.
- WebLogic SNMP MIB Reference at `http://e-docs.bea.com/wls/docs70/snmp/index.html`.

Logs

Many WebLogic Server operations generate logs of their activity. Each server has its own log as well as a standard HTTP access log. These log files can be configured and used in a variety of ways to monitor the health and activity of your servers and applications.

For more information, see:

- "Using Log Messages to Manage WebLogic Server" on page 4-1

- "Setting Up HTTP Access Logs" on page 6-15
- Using WebLogic Logging Services at http://e-docs.bea.com/wls/docs70/logging/index.html.

You can also configure a special domain log that contains a definable subset of log messages from all WebLogic Server instances in a domain. You can modify which logged messages from a local server appear in the domain log using the system administration tools. You can view this domain log using the Administration Console or a text editor/viewer.

For more information, see Domain Log Filters at `http://e-docs.bea.com/wls/docs70/ConsoleHelp/domain_log_filters.html`.

Editing config.xml

You can manage a WebLogic Server domain by manually editing the persistent store for configuration, the `config.xml`. (Other system administration tools automatically save the configuration to the `config.xml` file.) Because of the difficulty of correctly editing the XML syntax required in this file, this method of configuration is not recommended but may provide advantages in limited situations.

Note: Do not edit the `config.xml` file while the Administration Server is running.

For more information, see BEA WebLogic Server Configuration Reference at `http://e-docs.bea.com/wls/docs70/config_xml/index.html`.

Resources You Can Manage in a WebLogic Server Domain

This section discusses the domain resources you manage with the system administration tools.

Servers

The administrative concept of a *server* represents an instance of WebLogic Server in your domain. Using the system administration tools you can:

- Start and stop servers. (To start and stop servers on a remote machine, you must have Node Manager installed on the remote machine.) For more information see "Node Manager" on page 1-12.
- Configure a server's connections: ports, HTTP settings, jCom settings, and time outs.
- Configure HTTP server functionality and Virtual Hosts
- Configure logging and view logs
- Deploy applications to specific servers
- Configure WebLogic Server resources active on the server, such as JDBC Connection Pools and startup classes.

Clusters

WebLogic Server clusters allow you to distribute the work load of your application across multiple WebLogic Server instances. Clusters can improve performance and provide fail-over should a server instance become unavailable. For example, clusters provide several ways to replicate objects used in your applications so that data is not lost in the event of hardware failure.

You can architect combinations of clusters to distribute the work load in a way that provides the best performance for your applications.

Some services that are hosted on a single instance of WebLogic Server can be migrated from one server to another in the event of server failure. The system administration tools allow you to control these migrations.

For more information, see Using WebLogic Server Clusters at `http://e-docs.bea.com/wls/docs70/cluster/index.html`.

Machines

The administrative concept of a *machine* represents the physical machine that hosts an instance of WebLogic Server. WebLogic Server uses machine names to determine the optimum server in a cluster to which certain tasks, such as HTTP session replication, are delegated.

Using the system administration tools you can define one or more machines, configure Node Manger for those machines, and assign servers to the machines. For UNIX machines, you can configure UID and GID information.

For more information, see Using WebLogic Server Clusters at `http://e-docs.bea.com/wls/docs70/cluster/setup.html`.

Network Channels

Network channels are an optional feature that you can use to configure additional ports with one or more WebLogic Server instances or clusters. All servers and clusters that use a network channel inherit the basic port configuration of the channel itself. You can also customize a channel's port settings on an individual server using channel fine-tuning. This fine-tuning process creates an additional network resource called a Network Access Point.

For more information, see Configuring Network Resources at `http://e-docs.bea.com/wls/docs70/admin_domain/network.html`.

JDBC

Java Database Connectivity (JDBC) allows Java programs to interact with common DBMSs such as Oracle, Microsoft SQL Server, Sybase, and others.

Using the System Administration tools you can manage and monitor connectivity between WebLogic Server and your database management system. Connectivity is usually established through connection pools.

For more information, see "Managing JDBC Connectivity" on page 8-1.

JMS

The Java Message Service (JMS) is a standard API for accessing enterprise messaging systems that allow communication between applications.

Using the system administration tools, you can define configuration attributes to:

- Enable JMS
- Create JMS servers
- Create and/or customize values for JMS servers, connection factories, destinations (physical queues and topics), distributed destinations (sets of physical queue and topic members within a cluster), destination templates, destination sort order (using destination keys), persistent stores, paging stores, session pools, and connection consumers.
- Set up custom JMS applications.
- Define thresholds and quotas.
- Enable any desired JMS features, such as server clustering, concurrent message processing, destination sort ordering, persistent messaging, message paging, flow control, and load balancing for distributed destinations.

For more information, see "Managing JMS" on page 9-1.

WebLogic Messaging Bridge

A *Messaging Bridge* transfers messages between two messaging providers. The providers may be another implementation of WebLogic JMS or a 3rd party JMS provider.

For more information, see "Using the WebLogic Messaging Bridge" on page 10-1.

Web Servers and Web Components

WebLogic Server can perform as a fully functional Web server. WebLogic Server can server both static files such as HTML files and dynamic files such as Java servlets or Java ServerPages (JSP). Virtual hosting is also supported.

For more information on managing Web server functionality in WebLogic Server, see "Configuring WebLogic Server Web Components" on page 6-1.

Applications

Application deployment tools, including the Administration Console allow you to deploy, manage, update, and monitor your applications. The application deployment tools also allow you to deploy and update applications in a cluster of WebLogic Servers.

WebLogic Server 7.0 includes a new, two-phase deployment model that gives you more control over the deployment process. For more information, see WebLogic Server Deployment at `http://e-docs.bea.com/wls/docs70/programming/deploying.html`.

Using the system administration tools you can:

- Deploy applications to one or more WebLogic Servers or clusters in a domain.
- Configure runtime parameters for the applications.
- Monitor application performance
- Configure security parameters
- Protect access to an application based on security roles or a security policy. For more information see Setting Protections for WebLogic Resources at `http://e-docs.bea.com/wls/docs70/ConsoleHelp/security_7x.html#securitypolicies`.

Application Formats

You deploy applications in one or more of the following J2EE application formats:

- Web Applications
- Enterprise JavaBeans (EJB)
- Enterprise Applications
- J2EE Connectors
- Web Services. Web services are deployed as a Web Application that includes a special deployment descriptor that configures the Web Service.

For more information, see:

- WebLogic Server Deployment
- WebLogic Server Application Packaging
- Assembling and Configuring Web Applications
- "Deploying Applications" on page 5-1
- Developing WebLogic Server Applications
- Programming WebLogic Enterprise Java Beans
- Programming WebLogic J2EE Connectors
- Programming WebLogic Web Services
- Defining a Security Policy
- Setting Protections for WebLogic Resources

Editing Deployment Descriptors Using the Administration Console

You can use the Administration Console to edit the deployment descriptors for your J2EE applications. For more information, see:

- Application Deployment Descriptor Editor at `http://e-docs.bea.com/wls/docs70/ConsoleHelp/application_dde.html`
- Resource Adaptor Deployment Descriptor at `http://e-docs.bea.com/wls/docs70/ConsoleHelp/connector_dde.html`
- EJB Deployment Descriptor Editor at http://e-docs.bea.com/wls/docs70/ConsoleHelp/ejb_dde.html

- Web Application Deployment Descriptor Editor at `http://e-docs.bea.com/wls/docs70/ConsoleHelp/web_application_dde.html`

Editing and Creating Deployment Descriptors with WebLogic Builder

In addition to using the Administration Console to edit deployment descriptors you can also use the more robust WebLogic Builder tool that is included with your WebLogic Server distribution. WebLogic Builder is a stand-alone graphical tool for assembling a J2EE application, creating and editing deployment descriptors, and deploying an application on WebLogic Server. For more information, see WebLogic Builder Online Help at `http://e-docs.bea.com/wls/docs70/wlbuilder/index.html`.

Startup and Shutdown Classes

A startup class is a Java program that is automatically loaded and executed when a WebLogic Server is started or restarted and after other server initialization tasks have completed. A shutdown class is automatically loaded and executed when a WebLogic Server is shut down either from the Administration Console or using the `weblogic.Admin` shutdown command.

You use the system administration tools to register and manage startup and shutdown classes.

For more information, see "Starting and Stopping WebLogic Servers" on page 2-1.

JNDI

The Java Naming and Directory Interface (JNDI) API enables applications to look up objects—such as Data Sources, EJBs, JMS, and MailSessions—by name. You can view the JNDI tree through the Administration Console.

For additional information, see:

- "Managing JNDI" on page 11-1
- Programming WebLogic JNDI at http://e-docs.bea.com/wls/docs70/jndi/index.html.

Transactions

You use the system administration tools to configure and enable the WebLogic Server Java Transaction API (JTA). The transaction configuration process involves configuring:

- Transaction time outs and limits
- Transaction Manager behavior

For more information, see:

- "Managing Transactions" on page 7-1
- Programming WebLogic JTA

XML

The XML Registry is a facility for configuring and administering the XML resources of an instance of WebLogic Server. XML resources in WebLogic Server include the parser used by an application to parse XML data, the transformer used by an application to transform XML data, external entity resolution, and caching of external entities.

For more information, see Administering WebLogic Server XML at `http://e-docs.bea.com/wls/docs70/xml/xml_admin.html`.

Security

Security functionality has been completely re-written for WebLogic Server version 7. The new security system allows you to plug in third-party security solutions and also provides out-of-the box implementations for many common security systems. You can also create your own security solution and implement it in WebLogic Server.

For backwards compatibility, the security functionality available in version 6.0 and 6.1 of WebLogic Server is also supported when running in Compatibility Mode.

Using the administration tools, you can define realms, users, groups, passwords, ACLs and other security features.

For more information, see:

- Managing WebLogic Security at `http://e-docs.bea.com/wls/docs70/secmanage/index.html`.
- Using Compatibility Security in *Managing WebLogic Security* at `http://e-docs.bea.com/wls/docs70/secmanage/security6.html`.
- "Security" in the Administration Console Help at `http://e-docs.bea.com/wls/docs70/ConsoleHelp/security_7x.html`.
- Security Index Page

WebLogic Tuxedo Connector

WebLogic Tuxedo Connector provides interoperability between WebLogic Server applications and Tuxedo services. The connector allows WebLogic Server clients to invoke Tuxedo services and Tuxedo clients to invoke WebLogic Server Enterprise Java Beans (EJBs) in response to a service request.

For more information see WebLogic Tuxedo Connector at `http://e-docs.bea.com/wls/docs70/wtc.html`.

Jolt

Jolt is a Java-based client API that manages requests to BEA Tuxedo services via a Jolt Service Listener (JSL) running on a Tuxedo server.

For more information, see BEA Jolt at `http://e-docs.bea.com/tuxedo/tux80/interm/jolt.htm`.

Mail

WebLogic Server includes the JavaMail API version 1.1.3 reference implementation from Sun Microsystems.

For more information, see “Using JavaMail with WebLogic Server Applications” under Programming Topics at `http://e-docs.bea.com/wls/docs70/programming/topics.html`.

Starting and Using the Administration Console

This section contains instructions for starting and using the Administration Console.

Browser Support for the Administration Console

To run the Administration Console, use one of the following Web browsers:

- Microsoft Internet Explorer, version 5 on Windows
- Microsoft Internet Explorer, version 6 on Windows
- Netscape, version 4.7 on Windows or SunOS
- Netscape, version 6, on Windows or SunOS

If you use a Web browser that is not on the above list you may experience functional or formatting problems.

Starting the Administration Console

1. Start a WebLogic Administration Server. For more information, see “Starting an Administration Server” on page 2-12.

2. Open one of the supported Web browsers and open the following URL:

   `http://`*`hostname`*`:`*`port`*`/console`

   Where *`hostname`* is the DNS name or IP address of the Administration Server and *`port`* is the address of the port on which the Administration Server is listening for requests (7001 by default). If you started the Administration Server using Secure Socket Layer (SSL), you must add `s` after `http` as follows:

   `https://`*`hostname`*`:`*`port`*`/console`

   For more information about setting up SSL for system administration, see Server --> Connections --> SSL Ports at `http://e-docs.bea.com/wls/docs70/ConsoleHelp/domain_server_connections_ssl-ports.html`.

3. When the login page appears, enter the user name and the password you used to start the Administration Server (you may have specified this user name and password during the installation process) or enter a user name that belongs to one of the following security groups: Administrators, Operators, Deployers, or Monitors. These groups provide various levels of access to system administration functions in the Administration Console. For more information, see "Protecting System Administration Operations" on page 3-1.

   Using the security system, you can add or delete users to one of these groups to provide controlled access to the console. For more information, see "Protecting System Administration Operations" on page 3-1.

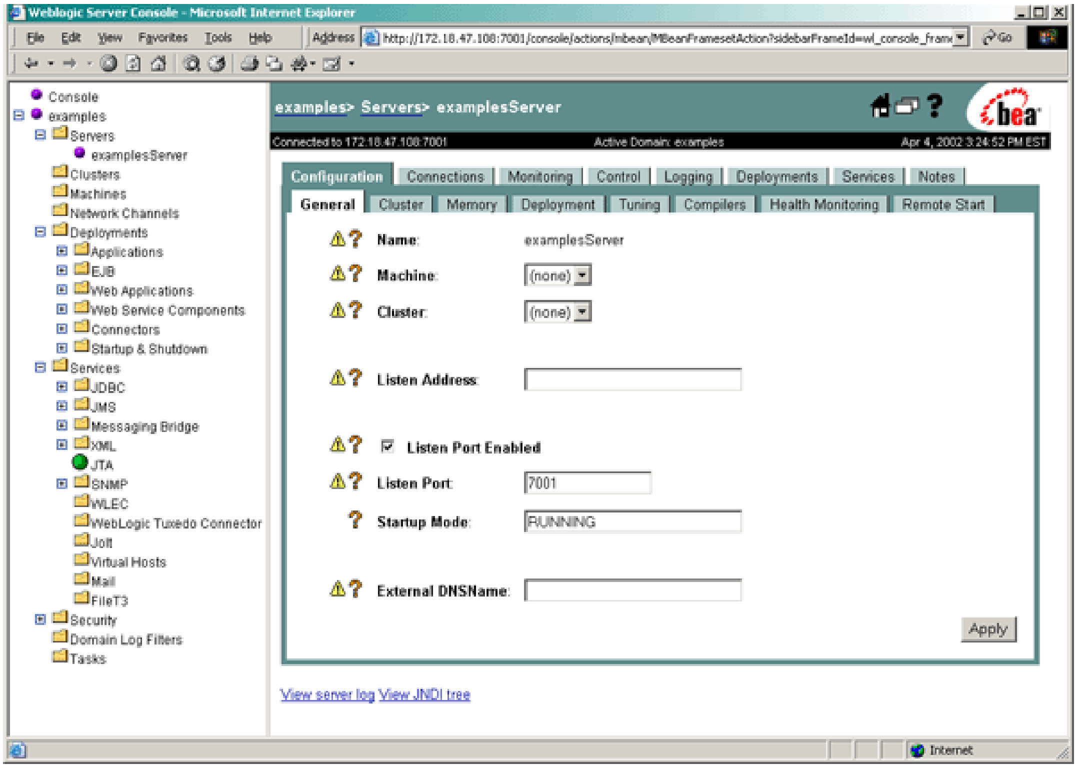
Note: If you have your browser configured to send HTTP requests to a proxy server, then you may need to configure your browser to not send Administration Server HTTP requests to the proxy. If the Administration Server is on the same machine as the browser, then ensure that requests sent to `localhost` or `127.0.0.1` are not sent to the proxy.

Using the Administration Console

This section provides instructions for using the Administration Console to manage and monitor a WebLogic Server domain.

Navigating in the Administration Console

Figure 1-2 Administration Console



Weblogic Server Console - Microsoft Internet Explorer

http://172.18.47.108:7001/console/actions/mbean/MBeanFramesetAction?sidebarFrameId=wl_console_fram

- Console
- examples
  - Servers
    - examplesServer
  - Clusters
  - Machines
  - Network Channels
  - Deployments
    - Applications
    - EJB
    - Web Applications
    - Web Service Components
    - Connectors
    - Startup & Shutdown
  - Services
    - JDBC
    - JMS
    - Messaging Bridge
    - XML
    - JTA
    - SNMP
    - WLEC
    - WebLogic Tuxedo Connector
    - Jolt
    - Virtual Hosts
    - Mail
    - FileT3
  - Security
  - Domain Log Filters
  - Tasks

examples> Servers> examplesServer

Connected to 172.18.47.108:7001 Active Domain: examples Apr 4, 2002 3:24:52 PM EST

Name:	examplesServer
Machine:	(none)
Cluster:	(none)
Listen Address:	
☑ Listen Port Enabled	
Listen Port:	7001
Startup Mode:	RUNNING
External DNSName:	

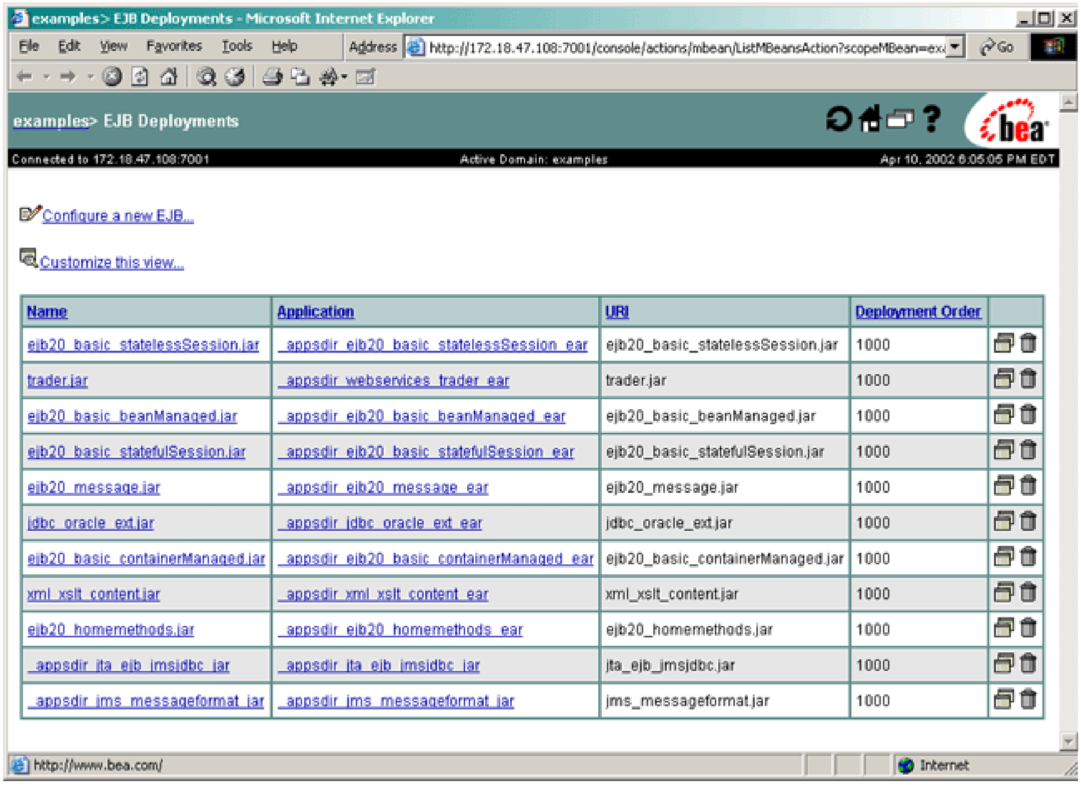
The left pane in the Administration Console contains a navigation tree that you use to navigate to tables of data, configuration pages, monitoring pages, or log files. By selecting (left-clicking) a node in the domain tree, you can display a table of data for a resource or configuration and monitoring pages for a selected resource. If a node in the tree is preceded by a plus sign, you can click on the plus sign to expand the tree to access additional resources.

A variety of operations are also accessible by right clicking on a node.

Once you select a node from the navigation tree, in the right pane you will either see a tabular listing of configured resources or objects or a tabbed interface.

When the data displayed is a table of data about resources or objects of a particular type, you can customize the table by adding or subtracting columns. You can also sort the data tables by clicking on the column headers. To customize a table, click on the Customize this view link at the top of the table.

Figure 1-3 Administration Console Table Page


examples> EJB Deployments - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://172.18.47.108:7001/console/actions/mbean/ListMBeansAction?scopeMBean=ex
Go
examples> EJB Deployments
bea
Connected to 172.18.47.108:7001
Active Domain: examples
Apr 10, 2002 6:05:05 PM EDT
Configure a new EJB...
Customize this view...
http://www.bea.com/
Internet

Name	Application	URI	Deployment Order	
ejb20_basic_statelessSession.jar	_appsdir_ejb20_basic_statelessSession_ear	ejb20_basic_statelessSession.jar	1000	
trader.jar	_appsdir_webservices_trader_ear	trader.jar	1000	
ejb20_basic_beanManaged.jar	_appsdir_ejb20_basic_beanManaged_ear	ejb20_basic_beanManaged.jar	1000	
ejb20_basic_statefulSession.jar	_appsdir_ejb20_basic_statefulSession_ear	ejb20_basic_statefulSession.jar	1000	
ejb20_message.jar	_appsdir_ejb20_message_ear	ejb20_message.jar	1000	
jdbc_oracle_ext.jar	_appsdir_jdbc_oracle_ext_ear	jdbc_oracle_ext.jar	1000	
ejb20_basic_containerManaged.jar	_appsdir_ejb20_basic_containerManaged_ear	ejb20_basic_containerManaged.jar	1000	
xml_xslt_content.jar	_appsdir_xml_xslt_content_ear	xml_xslt_content.jar	1000	
ejb20_homemethods.jar	_appsdir_ejb20_homemethods_ear	ejb20_homemethods.jar	1000	
_appsdir_jta_ejb_jmsjdbc_jar	_appsdir_jta_ejb_jmsjdbc_jar	jta_ejb_jmsjdbc.jar	1000	
_appsdir_jms_messageformat_jar	_appsdir_jms_messageformat_jar	jms_messageformat.jar	1000	

Configuring Objects or Resources

To configure the object or resource, click on its name. A tabbed screen will appear in the right panel that you can use to navigate through configuration or monitoring screens for the resource or object.

To edit your configuration, change values of the fields displayed in the right pane. After you edit the configuration, click the Apply button to execute the change and persist it to the `config.xml` file. Fields labeled with the ⚠ icon require you to restart any servers affected by the change before the change goes into effect.

Using the Administration Console to Manage Multiple Domains

Because an Administration Server can manage only a single active domain, you can access only that domain using the Administration Console. If you have separate Administration Servers running, each with its own active domain, you can switch from managing one domain to the other only by invoking the URL of the Administration Console on the Administration Server that you want to access.

For more information, see About the Administration Console at `http://e-docs.bea.com/wls/docs70/ConsoleHelp/console.html`.

Monitoring a Domain Using the Administration Console

To monitor a domain resource either right click on the resource in the navigation tree and select a monitoring option, or navigate to the resource and select the monitoring tab from the right pane. The data displayed represents the current state of the resource.

To update the information click the ↻ icon in the upper right section of the screen. The data will refresh regularly until you click the icon again. The icon displays a circular animation to indicate when auto-refresh is active. By default, the data refreshes every 10 seconds. You can alter the refresh interval by selecting the Console node and changing the value of **Auto-refresh every** field.

Monitoring Administration Console Tasks

You can monitor the progress of many operations you initiate from the console by clicking the Tasks node in the navigation tree of the Administration Console.

Getting Help for Using the Administration Console

Online help containing overviews, procedures, and information about configuration attributes is always available from the Administration Console. You can access the online help by clicking one of the following icons:

■Clicking this icon, located in the upper right corner of the console opens another browser window containing information about the console page you are viewing. You can also browse to other Administration Console topics from this window.

■ Clicking this icon, located next to a field, opens a small browser window containing information about that field.

Using WebLogic Server with Web Servers

You can proxy requests from popular Web servers to an instance of WebLogic Server or a cluster of WebLogic Servers by using one of the Web server plug-ins. Plug-ins are available for the following Web servers:

- Netscape Enterprise Server or IPlanet
- Microsoft Internet Information Server
- Apache

Because these plug-ins operate in the native environment of the Web server, management of the plug-ins is done using the administration facilities of that Web server.

For more information, see Using WebLogic Server with Plug-ins at `http://e-docs.bea.com/wls/docs70/plugins/index.html`.

Special servlets are also available to proxy requests from an instance of WebLogic Server to another instance of WebLogic Server or to a cluster of WebLogic Servers. For more information, see:

- Configure Proxy Plug-ins at `http://e-docs.bea.com/wls/docs70/plugins/http_proxy.html`.
- Proxying Requests to a WebLogic Cluster at `http://e-docs.bea.com/wls/docs70/cluster/setup.html#proxyplugins`.

Monitoring

The system administration tools contain extensive capabilities for monitoring WebLogic Servers, domains, and resources. Using the tools you can monitor:

- Server health and performance:
  - Execute Queues
  - Connections
  - Sockets
  - Threads
  - Throughput
  - Memory Usage
- Security:
  - Locked-out users
  - Invalid Logins
  - Login attempts
- Transactions:
  - Committed transactions
  - Rolledback transactions
- JMS connections and servers
- WebLogic Messaging Bridge
- Applications:
  - Servlet sessions
  - Connector connection pools
  - EJB performance
- JDBC connections and connection pools

For more information, see Monitoring a WebLogic Server Domain at `http://e-docs.bea.com/wls/docs70/admin_domain/monitoring.html`

Licenses

WebLogic Server requires a valid license to function.

An evaluation copy of WebLogic Server is enabled for 30 days so you can start using WebLogic Server immediately. To use WebLogic Server beyond the 30-day evaluation period, you will need to contact your salesperson about further evaluation or purchasing a license for each IP address on which you intend to use WebLogic Server. All WebLogic Server evaluation products are licensed for use on a single server with access allowed from up to five connections to the server.

If you downloaded WebLogic Server from the BEA Web site, your evaluation license is included with the distribution. The WebLogic Server installation program allows you to specify the location of the BEA home directory, and installs a BEA license file, `license.bea`, in that directory.

For more information, see "Managing WebLogic Server Licenses" on page 13-1

CHAPTER

2 Starting and Stopping WebLogic Servers

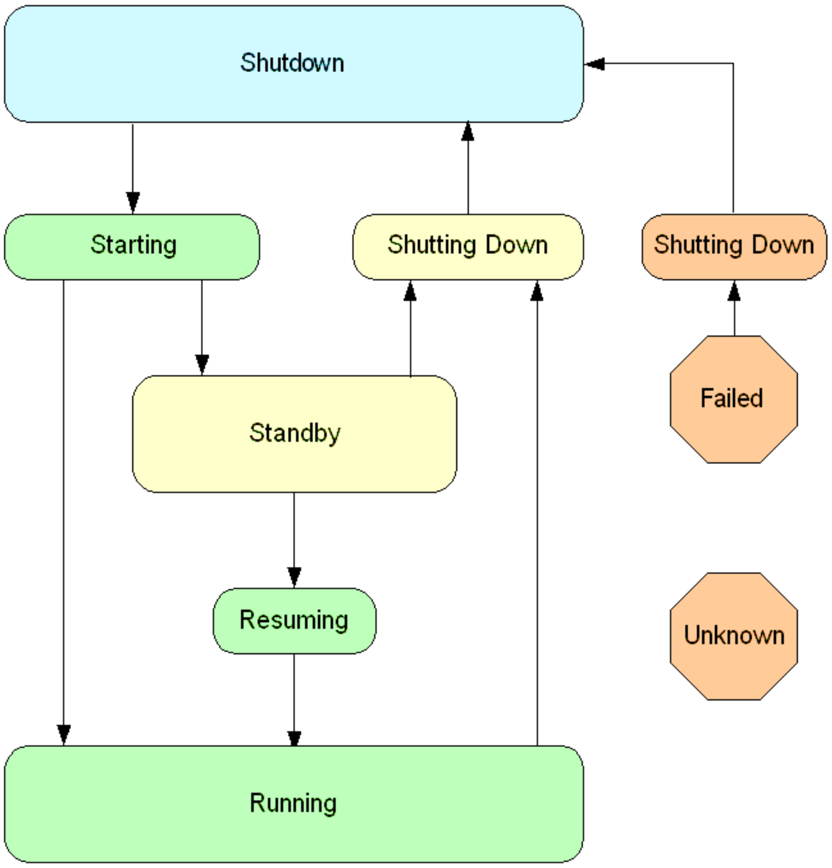
The following sections describe procedures for starting and stopping Administration Servers and Managed Servers:

- The Server Lifecycle
- Providing Usernames and Passwords to Start a Server
- Starting an Administration Server
- Starting a Managed Server
- Shutting Down WebLogic Servers
- Configuring Startup and Shutdown Classes
- Setting Up a WebLogic Server Instance as a Windows Service

The Server Lifecycle

A WebLogic Server can be in one of several states at any given time, and it follows a set of rules that determine how and when it can transition between those states. The series of states through which a server transitions is called the server lifecycle. (See Figure 2-1.)

Figure 2-1 The Server Lifecycle



The most common pattern of transitions is as follows:

1. `SHUTDOWN`. In this state, the server is configured but inactive.
2. `STARTING`. When you start a server, it takes the following actions:
   a. Retrieves its configuration data.

      An Administration Server retrieves the configuration data (including security configuration data) from the domain's configuration files. A Managed Server contacts the Administration Server for its configuration and security data. If you set up SSL, a Managed Server uses its own set of certificate files, key files, and other SSL-related files and contacts the Administration Server for the remaining configuration and security data.

b. Starts its kernel-level services, which include logging and timer services.

c. Initializes subsystem-level services with the configuration data that it retrieved in step 2a. These services include the following:

■ Security Service	■ JCA Container
■ RMI Service	■ JDBC Container
■ Cluster Service	■ EJB Container
■ IIOP Service	■ Web Container
■ Naming Service	■ Deployment Manager
■ RMI Naming Service	■ JMS Provider
■ File Service	■ Remote Management
	■ Transaction Service

d. If you have configured a server to use a separate administration port, the server enables remote configuration and monitoring. For information about administration ports, refer to Configuring a Domain-Wide Administration Port in the *Creating and Configuring WebLogic Server Domains* Guide.

e. Deploys modules in the appropriate container and in the order that you specify in the WebLogic Server Administration Console.

   For any startup classes that are configured to load before application deployments, the classes are loaded and run before the server deploys JDBC connection pools, Web applications, and EJBs.

f. For any startup classes that are configured to load after application deployments, the classes are loaded and run after the server deploys JDBC connection pools, Web applications, and EJBs.

3. `STANDBY`. (Available only if you have configured an administration port.) You can issue a command that starts a server and places it in this state. In this state, a server has initialized all of its services and applications and can accept administration commands and participate in cluster communication. It is not accessible for requests that come from external clients.

   A typical use of the `STANDBY` state is to keep a server available as a “hot” backup, especially in a high-availability or mission-critical environment. When you need to use the backup server, you can quickly resume its ability to process client requests.

4. `RUNNING`. In this state, a server offers its services to clients and can operate as a full member of a cluster.

5. `SHUTDOWN`. You can move a server into this state either from the `RUNNING` state or the `STANDBY` state. As it transitions to `SHUTDOWN`, a server goes through the `SHUTTING_DOWN` state.

   When you issue a graceful shutdown, the server invokes any shutdown classes that you have configured. You can shut down a server gracefully only from the `RUNNING` or `STANDBY` states.

   When you issue a forceful shutdown, the server notifies all applications and subsystems to drop all current work and release all resources. A forceful shutdown could result in rolled back transactions and session loss for some clients. You can shut down a server forcefully from any state.

A server can be in two additional states:

- `FAILED`. If one or more critical services become dysfunctional during the lifetime of server, the server transitions to the `FAILED` state. Your only option to recover from the `FAILED` state is to shut down the server. You can set up a server to restart itself if critical services become dysfunctional. For information about automatic restarts, refer to Server Self-Health Monitoring in the *Configuring and Managing Domains* Guide.
- `UNKNOWN`. If a server cannot be contacted, it is considered to be in the `UNKNOWN` state.

Controlling the Server Lifecycle

You can use any of the following interfaces to control a server's lifecycle:

- The Administration Console provides the following ways to control a server's lifecycle:
  - On the Server→Configuration→General tab, the Startup Mode field determines whether a server starts in `STANDBY` or `RUNNING` by default.
  - On the Server→Tuning tab, the Timeout for Server Lifecycle Operations field determines the number of seconds a lifecycle operation waits before timing

out. For more information, refer to "Timeout Period for LifeCycle Operations" on page 2-5.

- On the Server→Control→Start/Stop tab, a list of commands start, stop, and resume servers. For more information, refer to server tasks in the Administration Console Online Help.

- The `weblogic.Server` startup command includes an argument that overrides the default startup state. For information about the `-Dweblogic.management.startupMode=STANDBY` argument, refer to "Frequently Used Optional Arguments" on page 2-20.

- The `weblogic.Admin` utility provides the following commands:
  - START (requires the Node Manager)
  - STARTINSTANDBY (requires the Node Manager)
  - RESUME
  - SHUTDOWN
  - FORCESHUTDOWN

  An additional command, GETSTATE, returns the current state of a server.

  For information about using the `weblogic.Admin` utility, refer to Appendix B, "WebLogic Server Command-Line Interface Reference," or enter the following command at a command line:

  ```
  java weblogic.Admin HELP
  ```

  For information about the Node Manager, refer to Managing Server Availability with Node Manager in the *Creating and Configuring WebLogic Server Domains* Guide.

Timeout Period for LifeCycle Operations

When you issue a lifecycle command, the server notifies subsystems and applications of the requests and waits a number of seconds for the subsystems and application to respond. If they do not respond in the specified number of seconds, the server times out the lifecycle operation. The actions that it takes after the timeout depend on the operation.

This timeout period applies only to the SHUTDOWN and FORCESHUTDOWN operations. If the operation does not complete within the configured period, one of the following occurs:

- If the state of the server at that time was `SHUTTING_DOWN` or if the operation was FORCESHUTDOWN, then the server shuts down automatically.
- Otherwise, a `ServerLifecycleException` will be thrown with a message describing the timeout condition.

You can change the default timeout period on the Server→Tuning tab. For more information, refer to Setting the Timeout Period for LifeCycle Operations in the Administration Console Online Help.

Providing Usernames and Passwords to Start a Server

By default, a WebLogic Server prompts you to enter a username and password in the command shell that runs the server process. The username must belong to a role that is permitted to start servers. For information on roles and permissions, refer to "Protecting System Administration Operations" on page 3-1.

This section describes the following tasks:

- Specifying an Initial Administrative Username
- Bypassing the Prompt for Username and Password

Specifying an Initial Administrative Username

The Configuration Wizard prompts you to provide a username and password, which becomes the initial administrative username for the `myrealm` security realm. A **security realm** is a collection of components (providers) that authenticate usernames, determine the type of resources that the user can access, and provide other security-related services for WebLogic resources. WebLogic Server installs the `myrealm` security realm and uses it by default.

The first time you start a WebLogic Server, enter this initial administrative username and password. If you did not use the Configuration Wizard, the WebLogic Server prompts you to enter a initial username and password.

You can use the Administration Console to add users to `myrealm`. If you use an Authentication provider other than the one that WebLogic Server installs, you must use the provider's administration tools to create at least one user with administrative privileges. For information on granting administrative privileges, refer to "Protecting System Administration Operations" on page 3-1.

Note: The `guest` user is no longer supplied by default in WebLogic Server version 7.0. To use the `guest` user, you must run in Compatibility mode or define the `guest` user as a user in the Authentication provider for your security realm. For information about Compatibility mode, refer to Using Compatibility Security in the *Managing WebLogic Security* guide.

You can configure a WebLogic Server to use a different security realm. If you set up different security realms, you must designate one of those realms as the default. During its startup cycle, a WebLogic Server uses the default realm to authenticate the username that you supply.

Bypassing the Prompt for Username and Password

If you want to bypass the prompt for username and password, we recommend that you create and use a boot identify file, which contains your username and password in an encrypted format.

This section contains the following subsections:

- Creating a Boot Identity File for an Administration Server
- Creating a Boot Identity File for a Managed Server
- Using a Boot Identity File
- Removing a Boot Identity File After Startup
- Alternate Method: Passing Identity Information on the Command Line

Creating a Boot Identity File for an Administration Server

To create a boot identity file for an Administration Server:

1. Start the Administration Server at least once and provide the user credentials on the command line.

   During the Administration Server's initial startup process, it generates security files that must be in place before a server can use a boot identity file.

2. Place the following two lines in a text file:

   ```
   username=username
   password=password
   ```

   The username and password values must match an existing user account in the Authentication provider for the default security realm and must belong to a role that has permission to start a server. For information on roles and permissions, refer to "Protecting System Administration Operations" on page 3-1.

3. Save the file.

   If you save the file as `boot.properties` and locate it in the server's root directory, the server will automatically use this file during startup. For more information, refer to "Using a Boot Identity File."

The first time you use this file to start a sever, the server reads the file and then overwrites it with an encrypted version of the username and password.

Alternative Technique for Creating a Boot Identity File for an Administration Server

If you invoke the `weblogic.Server` class directly on the command line, instead of following the steps in the previous section, you can create a boot identity file by including the following options in the Java command:

```
-Dweblogic.management.username=username
-Dweblogic.management.password=password
-Dweblogic.system.StoreBootIdentity=true
```

These options cause the server instance to boot with the supplied user credentials and then store them in a file named `boot.properties`.

For example, the following command starts an Administration Server named myAdminServer and creates a boot identity file:

```
java -Dweblogic.management.username=username
-Dweblogic.management.password=password
-Dweblogic.system.StoreBootIdentity=true
-Dweblogic.Name=myAdminServer weblogic.Server
```

For more information about invoking the `weblogic.Server` class directly from a command line, refer to "Using the weblogic.Server Command" on page 2-17.

Note: If you use a script to start an Administration Server, BEA recommends that you do **not** use the technique described in this section for the following reasons:

- It requires you to store an unencrypted password in the startup script.
- Each time you run the script, the server boots with the supplied user credentials and then creates a new boot identity file.

Creating a Boot Identity File for a Managed Server

If a Managed Server uses the same root directory as the Administration Server, you do not need to create an additional boot identity file for the Managed Server, unless you want the Managed Server to run under different user credentials. For information about a server's root directory, refer to "A Server's Root Directory" on page 2-30.

In addition, if you use a Node Manager to start a Managed Server, you do not need to create a boot identity file. Instead, you must specify user credentials on the Managed Server's Remote Start tab in the Administration Console. For more information, refer to "Configure Startup Arguments for Managed Servers."

To create a boot identity file for a Managed Server:

1. Start the domain's Administration Server.
2. Copy the `SerializedSystemIni.dat` file from the Administration Server's root directory to the Managed Server's root directory.
3. Place the following two lines in a text file:

   ```
   username=username
   password=password
   ```

   The username and password values must match an existing user account in the Authentication provider for the default security realm and must belong to a role that has permission to start a server. For information on roles and permissions, refer to "Protecting System Administration Operations" on page 3-1.

4. Save the file.

   If you save the file as `boot.properties` and locate it in the server's root directory, the server will automatically use this file during startup. For more information, refer to "Using a Boot Identity File."

Using a Boot Identity File

A server instance uses a boot identity file as follows:

- If a server's root directory contains a valid `boot.properties`, it uses this file by default. For information about a server's root directory, refer to "A Server's Root Directory" on page 2-30.

- If you want to specify a different file (or if you do not want to store boot identity files in a server's root directory), you can include the following argument in the server's `weblogic.Server` startup command:

  ```
  -Dweblogic.system.BootIdentityFile=filename
  ```

  where *`filename`* is the fully qualified pathname of a valid boot identity file.

  If you use the `startWebLogic` script, add `-Dweblogic.system.BootIdentityFile` as a value of the `JAVA_OPTIONS` variable. For example:

  ```
  JAVA_OPTIONS=-Dweblogic.system.BootIdentityFile=C:\BEA\user_dom
  ains\mydomain\myidentity.prop
  ```

- If you do **not** want a server instance to use a boot identity file, include the following options in the server's `weblogic.Server` startup command:

  ```
  -Dweblogic.management.username=username
  -Dweblogic.management.password=password
  ```

  These options prevent a server instance from using any boot identity file and override other startup options that cause a server to use boot identity files.

  Note: If you use a script to start a server instance, BEA recommends that you do **not** use this technique because it requires you to store an unencrypted password in the startup script. Use this technique only if you invoke the `weblogic.Server` class directly from the command line. For more information, see "Using the weblogic.Server Command" on page 2-17.

- If a server is unable to access its boot identity file, it displays the username and password prompt in its command shell and writes a message to the log file.

For a given server instance, use only the boot identity file that the instance has created. WebLogic Server does not support copying a boot identity file from one server root directory to another.

For example, if you use ServerA to generate a boot identity file, use only that boot identity file with ServerA. Do not copy ServerA's boot identity file into the root directory of ServerB. Instead, create a boot identity file for ServerB as described in the previous sections.

Removing a Boot Identity File After Startup

If you want to remove the boot identity file after a server starts, you can include the following argument in the server's `weblogic.Server` startup command:

```
-Dweblogic.system.RemoveBootIdentity=true
```

This argument removes only the file that the server used to start. For example, if you specify `-Dweblogic.system.BootIdentityFile=c:\secure\boot.MyServer`, only `boot.MyServer` is removed, even if the server's root directory contains a file named `boot.properties`.

Alternate Method: Passing Identity Information on the Command Line

Using a boot identity file is the most secure and convenient way to bypass the interactive prompt. However, instead of using a boot identify file, you can add the following arguments to the `weblogic.Server` startup command:

```
-Dweblogic.management.username=username
-Dweblogic.management.password=password
```

If you supply both of these arguments, you can bypass the interactive prompt.

These options prevent a server instance from using any boot identity file and override other startup options that cause a server to use boot identity files.

Because the command to start a server can be long, typically you place most of the startup command in a script. Unless you are in an environment in which security is not a concern, we recommend that you do not save the `-Dweblogic.management.password=`*`password`* argument in a startup script.

For more information about these arguments, refer to "Using the weblogic.Server Command" on page 2-17.

Starting an Administration Server

A WebLogic Server runs as a process within a Java Virtual Machine (JVM). Each JVM can host only one server process. To start a server, you initiate a JVM with a set of arguments.

If a domain consists of only one WebLogic Server, that server is the Administration Server. If a domain consists of multiple WebLogic Servers, you must start the Administration Server before you start the Managed Servers.

The Administration Server and all Managed Servers in a domain must be the same WebLogic Server version. The Administration Server must be either at the same service-pack level or at a later service-pack level than the Managed Servers. For example, if the Managed Servers are at release 7.0, then the Administration Server can be either release 7.0 or 7.0 SP1. However, if the Managed Servers are at SP1, then the Administration Server must be at SP1. Each server within a domain must have a unique name.

This section describes starting an Administration Server by completing any of the following tasks from the local host:

- Starting an Administration Server from the Windows Start Menu
- Starting an Administration Server Using a Script
- Using the weblogic.Server Command
- Using the Default Configuration to Start a Server

For information, see:

- "Setting Up a WebLogic Server Instance as a Windows Service" on page 2-45
- "Binding to Protected Ports on UNIX" in the Administration Console Online Help

Note: When starting WebLogic Server, JDK 1.3 may throw an `OutOfMemory` error if you are trying to load a large number of classes. This error occurs even though there appears to be plenty of memory available. If you encounter a `java.lang.OutOfMemory` error exception when you start WebLogic Server, increase the value of the following JVM option:

```
java -XX:MaxPermSize=<value>
```

where <value> is some number in kilobytes.

For JDK1.3.1, the default value for `MaxPermSize` is 64m, where m stands for megabytes.

Starting an Administration Server from the Windows Start Menu

If you use the Configuration Wizard to create Single Server, an Administration Server with Managed Servers, or an Administration Server with Clustered Managed Servers on a Windows computer, the wizard prompts you to install the domain in the Windows Start Menu. If you choose yes, then you can do the following to start the Single Server or Administration Server:

From the Windows desktop, click Start→Programs→BEA WebLogic Platform 7.0→User Projects→*domain_name*→Start Server.

The Start Server command opens a command window and calls the *`domain_name`*`\startWebLogic.cmd` script, which is described in "Starting an Administration Server Using a Script" on page 2-14. When the server has successfully completed its startup process, it writes the following message to the command window:

```
<Notice> <WebLogicServer> <000360> <Server started in RUNNING mode>
```

Starting an Administration Server Using a Script

Because the arguments needed to start a WebLogic Server from the command line can be lengthy and prone to error, we recommend that you incorporate the command into a script.

This section describes the following tasks:

- Using the Configuration Wizard Scripts to Start an Administration Server
- Creating Your Own Script to Start an Administration Server
- Using a Non-Default JVM with WebLogic Server

Using the Configuration Wizard Scripts to Start an Administration Server

When you use the Configuration Wizard to create a domain, the wizard also creates a script that you can use to start an Administration Server for the domain. To use the script, enter one of the following commands at a command prompt:

- `*domain_name*\startWebLogic.cmd` (Windows)
- `*domain_name*\startWebLogic.sh` (UNIX and Windows. On Windows, this script supports the MKS and Cygnus BASH UNIX shell emulators.)

  where *domain_name* is the directory in which you located your domain.

The script sets values for some domain-specific variables and then calls the master startup script, *WL_HOME*\server\bin\startWLS.cmd (startWLS.sh on UNIX), where *WL_HOME* is the location in which you installed WebLogic Server. The master startup script sets environment variables, such as the location of the JVM, and then starts the JVM with WebLogic Server arguments.

Creating Your Own Script to Start an Administration Server

If you use some other means to create a domain (such as the Administration Console), you can create your own startup script that does the following:

1. Sets the value of a variable named SERVER_NAME. All servers in a domain must be named. For example,

```
set SERVER_NAME=myserver
```

In the domain's `config.xml` file, the name of a server is specified as `<Server Name=`*`serverName`*`>`. Make sure that the value for `set SERVER_NAME` refers to the server name as specified in `config.xml`.

2. Sets values for any of the following optional variables:

Table 2-1 Optional variables

Variable	Description
WLS_USER	Variable for setting a cleartext user for server startup. Instead of using this variable, we recommend that you use a boot identity file. For more information, refer to "Bypassing the Prompt for Username and Password" on page 2-7.
WLS_PW	Variable for setting a cleartext password for server startup. Instead of using this variable, we recommend that you use a boot identity file. For more information, refer to "Bypassing the Prompt for Username and Password" on page 2-7.
ADMIN_URL	If you specify a URL for this variable, the server will start as a Managed Server and will use the specified URL to contact its Administration Server. For more information, refer to "The Administration Server and Managed Servers" on page 1-6.
STARTMODE	Determines whether the server runs in production mode or development mode. Specify `true` for production mode servers or `false` for development mode. For more information on using production and development modes refer to "Development Mode vs. Production Mode" on page 2-28.
JAVA_OPTIONS	Java command-line options for running the server. The Java command-line options will be passed to the JVM after JAVA_VM and MEM_ARGS are passed. `-Dweblogic.ListenAddress` is an example of a Java option that you can call from the domain start script. For more information about command-line options, refer to "Using the weblogic.Server Command" on page 2-17. If you are listing multiple options in a UNIX shell, put quotes around the entire set of options and include spaces between each option. For example: `JAVA_OPTIONS="-Dweblogic.attribute=value -Djava.attribute=value"`

Table 2-1 Optional variables

Variable	Description
JAVA_VM	Java argument that specifies the mode in which the virtual machine runs. Use one of the following options: ■ `-server` ■ `-client` ■ `-hotspot` (Windows only) If you are using a JVM that does not support any of these operational modes, you must edit the master script to prevent these arguments from being passed to the JVM. For more information, refer to "Using a Non-Default JVM with WebLogic Server" on page 2-16.
MEM_ARGS	Variable to override the default memory arguments passed to Java. In the master start scripts, the options are set by default to `-Xms200m and -Xmx200m`.

3. Calls the master startup script, `*WL_HOME*\server\bin\startWLS.cmd` (`startWLS.sh` on UNIX).

   The master startup script sets environment variables, such as the location of the JVM, and then starts the JVM with WebLogic Server arguments. If you are not using the JVM installed with WebLogic Server, you must edit the master start script. For more information, refer to "Using a Non-Default JVM with WebLogic Server" on page 2-16.

4. If you plan to locate your startup script outside of the domain's root directory, your script must include the following value for the `JAVA_OPTIONS` variable:

   ```
   -Dweblogic.RootDirectory=path
   ```

   where *path* specifies the location of the domain's root directory.

   For example,

   ```
   JAVA_OPTIONS=-Dweblogic.RootDirectory=c:\serverRoot
   ```

Using a Non-Default JVM with WebLogic Server

If you are not using the JVM installed with WebLogic Server, you must edit the master start script so that the `JAVA_HOME` variable specifies the correct location of the JVM on your system.

For detailed information, see "Using a Non-Bundled JVM With WebLogic Platform" at the following URL: http://e-docs.bea.com/platform/docs70/relnotes/relnotes.html.

Using the weblogic.Server Command

`weblogic.Server` is the command that starts a WebLogic Server on a local host. The startup scripts described in previous sections are wrappers that send a consistent set of options to this command. While we recommend that you incorporate this command and its options into a startup script, for simple invocations you can enter the `weblogic.Server` command directly on the command line.

For example, a simple invocation for starting the examples server on Windows is as follows (you must enter this command from the *`WL_HOME`*`\samples\server\config\examples` directory):

```
c:\bea\jdk131\bin\java
-hotspot -Xms200m -Xmx200m
-classpath "c:\bea\jdk131\lib\tools.jar;
   c:\bea\weblogic700\server\lib\weblogic_sp.jar;
   c:\bea\weblogic700\server\lib\weblogic.jar;"
-Dweblogic.Name=examplesServer
-Dbea.home="C:\bea"
-Djava.security.policy="c:\bea\weblogic700\server\lib\weblogic.policy"
weblogic.Server
```

This section describes the following:

- Setting the Classpath
- Command Syntax for weblogic.Server
- Required Arguments
- Frequently Used Optional Arguments
- Other Optional Arguments
- Development Mode vs. Production Mode
- Startup Arguments for the Administration Port and the weblogic.Admin Utility
- A Server's Root Directory

For information about starting a Managed Server on a remote host, refer to Managing Server Availability with Node Manager in the *Creating and Configuring WebLogic Server Domains* Guide.

Setting the Classpath

The Java Virtual Machine (JVM) uses a setting called *classpath* to locate essential files and directories.

You can use the following command to set the classpath for a WebLogic Server:

`WL_HOME\server\bin\setWLSEnv.cmd` (on Windows)
`WL_HOME/server/bin/setWLSEnv.sh` (on UNIX)

Instead of using `setWLSEnv`, you can use an environment variable or a `-classpath` argument in the startup command. Regardless of the method you choose, include the following in the classpath for the JVM that runs your instances of WebLogic Server:

- `WL_HOME/server/lib/weblogic_sp.jar`

  Depending on which WebLogic Server release, service pack, or patch that you have installed, this file might not exist on your system. Regardless of whether the file currently exists on your system, we recommend that you include `WL_HOME/server/lib/weblogic_sp.jar` on your classpath to ensure compatibility with any updates. You must add this file to the classpath before you add `weblogic.jar`.

- `WL_HOME/server/lib/weblogic.jar`

- If you use the trial version of PointBase, an all-Java database management system, then include the following files:

  `SAMPLES_HOME/server/eval/pointbase/server/lib/pbserver41ev.jar` and `pbclient41ev.jar`

  where `SAMPLES_HOME` is `WL_HOME/samples`.

- If you use WebLogic Enterprise Connectivity, include the following files:

  `WL_HOME/server/lib/wlepool.jar`
  `WL_HOME/server/lib/wleorb.jar`

  where `WL_HOME` is the directory where you installed WebLogic Server.

Command Syntax for weblogic.Server

The syntax for the `weblogic.Server` command is as follows:

```
java RequiredArguments [OptionalArguments] weblogic.Server
```

Required Arguments

The following table describes the required arguments for starting a WebLogic Server from the `java` command line.

Table 2-2 Required Arguments for Starting a Server

Argument	Description
`-Xms` and `-Xmx`	Specify the minimum and maximum values (in megabytes) for Java heap memory. For example, you may want to start the server with a default allocation of 200 megabytes of Java heap memory to the WebLogic Server. To do so, you can start the server with the `java -Xms200m` and `-Xmx200m` options. For best performance it is recommended that the minimum and maximum values be the same so that the JVM does not resize the heap. The values assigned to these parameters can dramatically affect the performance of your WebLogic Server and are provided here only as general defaults. In a production environment you should carefully consider the correct memory heap size to use for your applications and environment.
`-classpath`	The minimum content for this option is described under "Setting the Classpath" on page 2-18. **Note:** This is optional if the `classpath` is set in the user environment.
`-Dweblogic.Name=` *`servername`*	Assigns a name to the server. Server names must be unique within a domain. For example, if you name a server instance `ManagedServer1` in a domain named `DomainA`, you cannot name another server instance `ManagedServer1` in `DomainA`.

Table 2-2 Required Arguments for Starting a Server

Argument	Description
`-Dbea.home=`*`bea_home`*	Specifies the location of the BEA home directory, which contains licensing and other essential information.

Frequently Used Optional Arguments

The following table describes optional arguments that are frequently used. The description of each argument indicates whether it can also be set through the Administration Console or some other WebLogic Server command. Any argument that sets an attribute for a Managed Bean (MBean) can also be set through the MBean's API. "Other Optional Arguments" on page 2-27, describes setting MBean attributes.

Table 2-3 Frequently Used Optional Arguments

Argument	Description
`-Dweblogic.RootDirectory=`*`path`*	Specifies the server's root directory. For more information, refer to "A Server's Root Directory" on page 2-30.

`-Dweblogic.ConfigFile=` *`file_name`*	Specifies a configuration file for your domain. The *`file_name`* value must refer to a valid XML file that conforms to the `config.dtd`. The XML file must exist in the Administration Server's root directory, which is either the current directory or the directory that you specify with `-Dweblogic.RootDirectory`.
	The *`file_name`* value cannot contain a pathname component. For example, the following value is invalid:
	`-Dweblogic.ConfigFile=c:\mydir\myfile.xml`
	Instead, use the following arguments:
	`-Dweblogic.RootDirectory=c:\mydir`
	`-Dweblogic.ConfigFile=myfile.xml`
	For information about `config.dtd`, refer to BEA WebLogic Server Configuration Reference.
	If you do not specify this value, the default is `config.xml`.

Table 2-3 Frequently Used Optional Arguments

Argument	Description
`-Dweblogic.management.` `username=`*`username`*	Specifies the username. The username must belong to a role that has permission to start a server. For information on roles and permissions, refer to "Protecting System Administration Operations" on page 3-1. This option prevents a server instance from using any boot identity file and overrides other startup options that cause a server to use boot identity files. For more information, refer to "Bypassing the Prompt for Username and Password" on page 2-7.
`-Dweblogic.management.` `password=`*`password`*	Specifies the user password. This option prevents a server instance from using any boot identity file and overrides other startup options that cause a server to use boot identity files. For more information, refer to "Bypassing the Prompt for Username and Password" on page 2-7.
`-Dweblogic.ListenAddress=`*`host`*	Specifies a listen address for this server. The *`host`* value must be either the DNS name or the IP address of the server. This option sets the value of the `listenAddress` attribute in the `ServerMBean`, which is also accessible from the Administration Console under Server→Configuration→General→Listen Address. If you do not specify a Listen Address, a server uses either the machine's DNS name or the IP address. We recommend that you specify a known IP address or DNS name and that you use the Administration Console instead of this argument to do so. For more information, refer to Configuring Network Resources.
`-Dweblogic.ListenPort=` *`portnumber`*	Enables and specifies the plain-text (non-SSL) listen port for this server. The argument sets the value of the `listenPort` attribute in the `ServerMBean`, which is also accessible from the Administration Console under Server→Configuration→General→Listen Port. If you do not specify a Listen Port, a server uses 7001 as the default. For more information, refer to Configuring Network Resources.

Table 2-3 Frequently Used Optional Arguments

Argument	Description
`-Dweblogic.ssl.ListenPort=`*`portnumber`*	Enables and specifies the port at which this WebLogic Server listens for SSL connection requests. The argument sets the value of the `listenPort` attribute in the `SSLMBean`, which is also accessible from the Administration Console under Server→Connections→SSL Ports→SSL Listen Port. If you do not specify a Listen Port, a server uses 7002 as the default. For more information, refer to Configuring Network Resources.
`-Dweblogic.system.StoreBootIdentity=true`	Creates a `boot.properties` in the server's root directory. The file contains the username and an encrypted version of the password that you used to start the server. For more information, refer to "Bypassing the Prompt for Username and Password" on page 2-7.
`-Dweblogic.system.BootIdentityFile=`*`filename`*	Specifies a boot identity file that contains a username and password. The *`filename`* value must be the fully qualified pathname of a valid boot identity file. For example: `-Dweblogic.system.BootIdentityFile=C:\BEA\wlserver7.0\user_config\mydomain\myidentity.prop` If you do not specify a filename, a server uses the `boot.properties` in the server's root directory. If there is no boot identity file, the server prompts you to enter a username and password.
`-Dweblogic.system.RemoveBootIdentity=true`	Removes the boot identity file after a server starts.
`-Dweblogic.management.pkpassword=`*`pkpassword`*	Specifies the password for retrieving Secure Socket Layer (SSL) private keys from an encrypted flat file. Use this option if you store private keys in an encrypted flat file.

Table 2-3 Frequently Used Optional Arguments

Argument	Description
`-Dweblogic.security.SSL.trustedCAKeyStore=`*`path`*	If you use SSL, you can use this argument to specify the certificate authorities that the server or client trusts. The *path* value must be a relative or qualified name to the Sun JKS keystore file (contains a repository of keys and certificates). If you do not specify this argument, the WebLogic Server or client trusts all of the certificates that are specified in *`JAVA_HOME`*`\jre\lib\security\cacerts`. We recommend that you do not use the demonstration certificate authorities in any type of production deployment.
`-Dweblogic.security.SSL.ignoreHostnameVerification=true`	Disables host-name verification. Include this argument if you want to use the demonstration digital certificates that are shipped with WebLogic Server. **Note:** BEA does not recommend using the demonstration digital certificates or turning off host name verification in a production deployment. If you do not specify this argument, the Host Name Verifier in WebLogic Server compares the SubjectDN of a digital certificate with the host name of the server that initiated the SSL connection. If the SubjectDN and the host name do not match, the SSL connection is dropped.
`-Dweblogic.security.SSL.HostnameVerifier=`*`hostnameverifierimplmentation`*	Specifies the name of a custom Host Name Verifier class. The class must implement the `weblogic.security.SSL.HostnameVerifier` interface.
`-Dweblogic.security.SSL.sessionCache.ttl=`*`sessionCacheTimeToLive`*	Modify the default server-session time-to-live for SSL session caching. The *`sessionCacheTimeToLive`* value specifies (in seconds) the session cache time-to-live. For `sessionCache.ttl`: ■ The minimum value is `1` ■ The maximum value is `Integer.MAX_VALUE` ■ The default value is `90`

Table 2-3 Frequently Used Optional Arguments

Argument	Description

`-Djava.security.manager`
`-Djava.security.policy=` *`filename`*
	Enable the Java 2 security manager, which prevents untrusted code from performing actions that are restricted by the policy file.
	The `-Djava.security.policy` argument specifies a filename (using a relative or fully-qualified pathname) that contains Java 2 security policies.
	The WebLogic Server sample policy file, which you can edit and use, is *`WL_HOME`*`\server\lib\weblogic.policy`. For more information, refer to Modifying the weblogic.policy File for General Use in the *Managing WebLogic Security* guide.

`-Dweblogic.security.anonymous UserName=guest`	Enables support for the `guest` user account. If you start a WebLogic Server instance with this argument, you must also add the `guest` user to the Authentication provider in the default security realm. For more information, refer to Creating Users in the *Securing WebLogic Resources* guide.
`-Dweblogic.management.commo. dumpFormat=<encrypted | cleartext>`	As described in *Recovering Failed Servers*: Backing Up Security Data in the *Creating and Configuring WebLogic Server Domains* guide, security domain configuration information is stored in the *`domain_name`*`/userConfig/Security` directory. You can back up the data in a security domain or export the data to a new security domain using `WebLogicMBeanLoader`, which outputs an XML file. By default, the values of password attributes are not printed in clear text in the XML file but instead are masked as `"*****"`. This avoids the security risk of printing passwords in clear text in the file system, but makes these files non-portable to different domains since the ***** mask strings cannot be intrepreted meaningfully. To make the XML file portable, you need to print clear text values to it, or else encrypted values that the new domain can decrypt. The value `encrypted` prints the attribute values in 3DES encrypted form. This should only be used when using the same `Encryption` salt in the new domain. The value `cleartext` prints the attribute values in clear text. Omitting this argument or using any other value obscures the attribute values, printing *****.

Table 2-3 Frequently Used Optional Arguments

Argument	Description
`-Dweblogic.management.startupMode=STANDBY`	Starts a server and places it in the `STANDBY` state. To use this startup argument, you must configure a server to use a separate administration port. For information about administration ports, refer to Configuring a Domain-Wide Administration Port in the *Creating and Configuring WebLogic Server Domains* Guide. This value overrides any `startupMode` value specified in the Administration Console under Server→Configuration→General tab for the current session only. If you do not specify this value (either on the command line or in `config.xml`), the default is to start in the `RUNNING` state.
`-Dweblogic.ProductionModeEnabled={true | false}`	Determines whether all servers in a domain start in production mode. This option is applicable only when starting the Administration Server. All Managed Servers start in the same mode as the Administration Server. A `true` value prevents a WebLogic Server from automatically deploying and updating applications that are in the `domain_name/applications` directory. If you do not specify this option, the assumed value is `false`. For more information, refer to "Development Mode vs. Production Mode" on page 2-28.

Table 2-3 Frequently Used Optional Arguments

Argument	Description

`-Dweblogic.management.`
`discover={true | false}`
	Determines whether an Administration Server recovers control of a domain after the server fails and is restarted.
	A `true` value causes an Administration Server to refer to its `running-managed-servers.xml` file, which contains information about the deployment state of deployable modules and a list of all Managed Servers that are currently running. When the Administration Server starts with this specified as `true`, it communicates with the Managed Servers and informs them that it is running.
	A `false` value prevents an Administration Server from referring to this file and thus prevents it from communicating with any Managed Servers that are currently active in the domain.
	Caution: Specify `false` for this option only in the development environment of a single server. Specifying `false` can cause server instances in the domain to have an inconsistent set of deployed modules.
	If you do not specify this option, the assumed value is `true`.

`-Dweblogic.Stdout="`*`filename`*`"`	Redirects the server and JVM's standard output stream to a file. You can specify a pathname that is fully qualified or relative to the WebLogic Server root directory. For more information, refer to "Redirecting System.out and System.err to a File" on page 4-11.
`-Dweblogic.Stderr="`*`filename`*`"`	Redirects the server and JVM's standard error stream to a file. You can specify a pathname that is fully qualified or relative to the WebLogic Server root directory. For more information, refer to "Redirecting System.out and System.err to a File" on page 4-11.

Table 2-3 Frequently Used Optional Arguments

Argument	Description
`-Dweblogic.AdministrationMBeanAuditingEnabled={true | false}`	Determines whether the Administration Server emits configuration auditing log messages when a user changes the configuration or invokes management operations on any resource within a domain. By default, the Administration Server does not emit configuration auditing messages. See "Configuration Auditing" on page 4-13.
`-Dweblogic.net.http.URLStreamHandlerFactory=`*`classname`*	Used to override the default WebLogic Server HTTP stream handler. To use this option, write a class that implements the `java.net.URLStreamHandlerFactory` interface. In addition to implementing the `createURLStreamHandler("http")` method specified by the interface, the class must include a `main()` method that calls `java.net.URL.setURLStreamHandlerFactory()` with an instance of the class as its argument. Set this system property to the name of this class. For more information, see the javadocs for the `java.net.URL.URLStreamHandlerFactory` interface.

Other Optional Arguments

You can use arguments of the `weblogic.Server` startup command to set attributes of the following MBeans:

- `ServerMBean`. Use the following syntax:
  `-Dweblogic.`*`attribute-name=value`*

  For example, to set the value of the `listenPort` attribute,
  `-Dweblogic.ListenPort=7010`

- `LogMBean`. Use the following syntax:
  `-Dweblogic.log.`*`attribute-name=value`*

  For example, to set the value of the `FileName` attribute,
  `-Dweblogic.log.FileName="C:\logfiles\myServer.log"`

- `SSLMBean`. Use the following syntax:
  `-Dweblogic.ssl.`*`attribute-name`*`=`*`value`*

  For example, to set the value of the `Enable` attribute to `true`,
  `-Dweblogic.ssl.Enable="true"`

You can set any attribute that the MBean exposes as a setter method. In the above syntax statements, *`attribute-name`* is the name of an MBean's setter method without the `set` prefix.

For example, the `ServerMBean` exposes its listen port attribute with the following setter method:

- `setListenPort()`

To set the listen port value from the `weblogic.Server` command, use the following syntax: `-Dweblogic.ListenPort=`*`portnumber`*

The command-line arguments override any settings currently in the MBean and they are not persisted to the domain's `config.xml` file.

Development Mode vs. Production Mode

You can run WebLogic Server in two different modes: development and production. You use development mode to test your applications. Once they are ready for a production environment, you deploy your applications on a server that is started in production mode.

Development mode enables a WebLogic Server to automatically deploy and update applications that are in the *`domain_name`*`/applications` directory (where *`domain_name`* is the name of a WebLogic Server domain).

Production mode disables the auto-deployment feature. Instead, you must use the WebLogic Server Administration Console or the `weblogic.Deployer` tool. For more information on deployment, refer to WebLogic Server Deployment in the *Developing WebLogic Server Applications* Guide.

By default, a WebLogic Server runs in development mode. To specify the mode for a server, do one of the following:

- If you use the `startWebLogic` startup script, edit the script and set the `STARTMODE` variable as follows:
  - `STARTMODE = false` enables deployment mode

  - `STARTMODE = true` enables production mode

  For more information about `startWebLogic`, refer to "Starting an Administration Server Using a Script" on page 2-14.

- If you start a server entering the `weblogic.Server` command directly on the command line, use the `-Dweblogic.ProductionModeEnabled` option as follows:
  - `-Dweblogic.ProductionModeEnabled=false` enables deployment mode
  - `-Dweblogic.ProductionModeEnabled=true` enables production mode

Note: Production or development mode is a domain-wide setting, which you specify when starting the Administration Server. All Managed Servers run in the same mode as the Administration Server.

Startup Arguments for the Administration Port and the weblogic.Admin Utility

An **administration port** is a separate port that you must set up if you want to start server instances in the `STANDBY` state or if you want to separate administration traffic from application traffic in your domain.

If you want to use an administration port to carry requests from the `weblogic.Admin` utility, you must do the following:

1. Set up SSL and an administration port for **all** server instances in the domain as described in "Configuring a Domain-Wide Administration Port" in the *Creating and Configuring WebLogic Server Domains* guide.
2. Include the following startup arguments in the `weblogic.Server` command for all server instances:

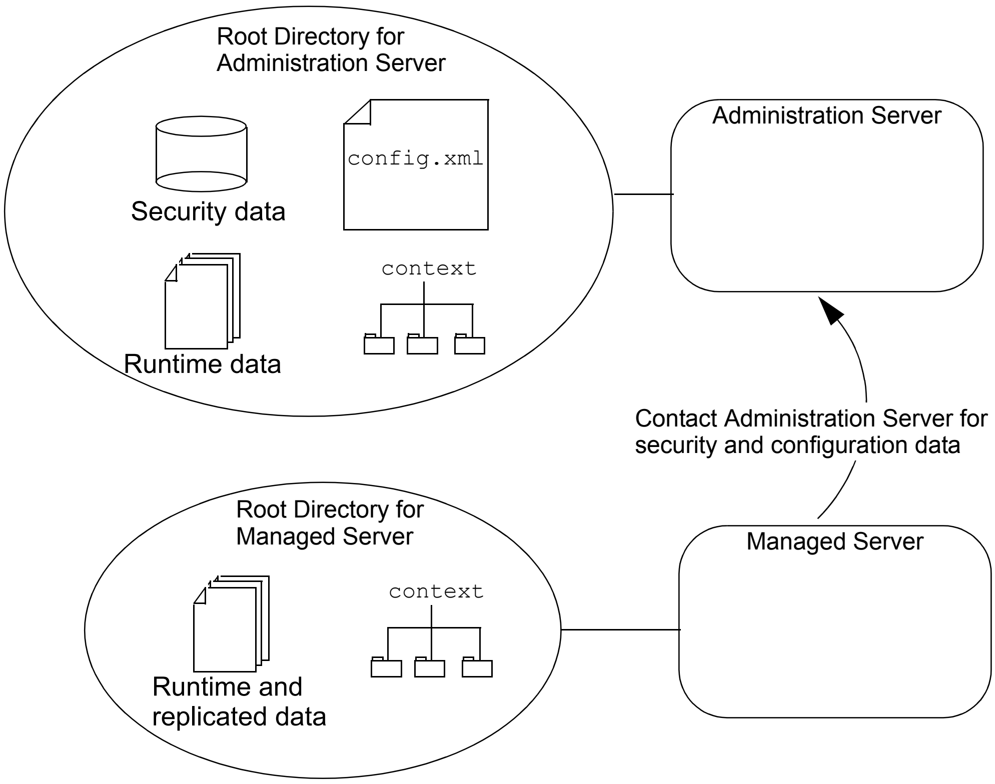
   ```
   -Dweblogic.security.SSL.trustedCAKeystore=path
   -Dweblogic.security.SSL.ignoreHostnameVerification=true
   ```

A Server's Root Directory

All instances of WebLogic Server use a root directory to store runtime data and to provide the context for any relative pathnames in the server's configuration.

In addition, an Administration Server uses its root directory as a repository for the domain's configuration data (such as `config.xml`) and security resources (such as the default, embedded LDAP server), while a Managed Server stores replicated administrative data in is root directory. (See Figure 2-2.)

Figure 2-2 Root Directory for WebLogic Server Instances


Root Directory for Administration Server
Security data
config.xml
Runtime data
context
Administration Server
Contact Administration Server for security and configuration data
Root Directory for Managed Server
Runtime and replicated data
context
Managed Server

Multiple instances of WebLogic Server can use the same root directory. However, if your server instances share a root directory, make sure that all relative filenames are unique. For example, if two servers share a directory and they both specify `.\MyLogFile`, then each server instance will overwrite the other's `.\MyLogFile`.

To determine the root directory for an Administration Server, WebLogic Server does the following:

- If the server's startup command includes the `-Dweblogic.RootDirectory=`*`path`* option, then the value of *`path`* is the root directory.

- If `-Dweblogic.RootDirectory=`*`path`* is not specified, and if the working directory (that is, the directory from which you issue the startup command) contains a `config.xml` file, then the working directory is the root directory.
- If neither of the previous statements is true, then the server looks for a `config.xml` file in *`working-directory`*`/config/`*`domain-name`*. If it finds `config.xml` in this directory, then *`working-directory`*`/config/`*`domain-name`* is the root directory.
- If WebLogic Server cannot find a `config.xml` file, then it offers to create one, as described in "Using the Default Configuration to Start a Server" on page 32.

To determine the root directory for a Managed Server, WebLogic Server does the following:

- If the server's startup command includes the `-Dweblogic.RootDirectory=`*`path`* option, then the value of *`path`* is the root directory.
- If `-Dweblogic.RootDirectory=`*`path`* is not specified, then the working directory is the root directory. For example, if you run the `weblogic.Server` command from `c:\config\MyManagedServer`, then `c:\config\MyManagedServer` is the root directory.

To make it easier to maintain your domain configurations and applications across upgrades of WebLogic Server software, it is recommended that the root directory not be the same as the installation directory for the WebLogic Server software.

Using the Default Configuration to Start a Server

If you run into problems in your environments and want to boot a server with a clean (default) configuration, you can start WebLogic Server in such a way that it generates a new `config.xml` file.

This new `config.xml` file contains only the default settings, unless you use command-line arguments to override the defaults. The username and password that you supply when you start the server becomes the default administrative user.

Note that the server starts as an Administration Server in a new domain. There are no other servers in this domain, nor are any of your deployments or third-party solutions included. You can add them as you would add them to any WebLogic domain.

To cause WebLogic Server to generate a new `config.xml` file, start an Administration Server using a server root directory that does not already contain a `config.xml` file. For example, you can do the following:

1. Make a new directory for your default configuration.
2. Navigate to that directory, and in a command shell, enter the following command:
   `WL_HOME\server\bin\setWLSEnv.cmd` (Windows)
   `WL_HOME/server/bin/setWLSEnv.sh` (UNIX)
3. Enter the following command:

   ```
   java weblogic.Server
   ```

4. When the server prompts you, enter a username and password. This will become the default administrative user for the domain.
5. When the server prompts you to create a new default configuration, enter `y`.

The server prompts you to reenter your password. Then it starts a server with the new configuration.

Starting a Managed Server

Before you can run a WebLogic Server as a Managed Server, you must do the following:

- Start the domain's Administration Server.
- Create an entry for that server in the configuration for the domain as described in Adding a Managed Server to a Domain.

After describing how to add a Managed Server to a domain, this section describes starting a Managed Sever by completing any of the following tasks:

- Starting a Managed Server from the Windows Start Menu
- Starting a Managed Server Using a Script
- Starting a Managed Server from the Command Line
- Configuring a Connection to the Administration Server

- Specifying the Default Startup State
- Starting a Remote Managed Server
- Starting and Killing All WebLogic Servers in a Domain or Cluster

For information on starting Managed Servers when the Administration Server is unavailable, refer to Starting a Managed Server When the Administration Server Is Not Available in the *Creating and Configuring WebLogic Server Domains* Guide.

Adding a Managed Server to a Domain

To add a Managed Server to a domain, do the following:

1. Start the Administration Server for the domain.
2. Invoke the Administration Console by pointing your browser at `http://`*`hostname`*`:`*`port`*`/console`, where *`hostname`* is the name of the machine where the Administration Server is running and *`port`* is the listen port number that you have configured for the Administration Server (default is 7001).
3. If the server runs on a machine that is different from the Administration Server's machine, do the following:
   a. In the left pane of the Administration Console, click the Machines node.
   b. In the right pane, click Configure a new Machine.
   c. Enter a name and click Create.
4. In the left pane, click the Servers node.
5. On the right pane, click Configure a new Server and do the following:
   a. Enter a name for the server.

      Within a given domain, each server name must be unique.
   b. If you created a machine, select it for this Managed Server.
   c. Click Create.

6. If you want to set up an administration channel for this server, refer to Configuring a Domain-Wide Administration Port in the *Creating and Configuring WebLogic Server Domains* Guide.

Starting a Managed Server from the Windows Start Menu

If you use the Configuration Wizard to create a Managed Server (with owning Administration Server configuration) on a Windows computer, the wizard prompts you to install the domain in the Windows Start Menu. If you choose yes, then you can do the following to start the Managed Server:

From the Windows desktop, click Start→Programs→BEA WebLogic Platform 7.0→User Projects→*domain_name*→Start Server.

The Start Server command opens a command window and calls the `domain_name\startManagedWebLogic.cmd` script, which is described in "Starting a Managed Server Using a Script" on page 2-35. When the server has successfully completed its startup process, it writes the following message to the command window:

```
<Notice> <WebLogicServer> <000360> <Server started in RUNNING mode>
```

Starting a Managed Server Using a Script

Because the arguments needed to start a WebLogic Server from the command line can be lengthy and prone to error, we recommend that you incorporate the command into a script.

This section describes the following tasks:

- Using the Configuration Wizard Scripts to Start a Managed Server
- Creating Your Own Script to Start a Managed Server

If you are not using the JVM installed with WebLogic Server, refer to "Using a Non-Default JVM with WebLogic Server" on page 2-16.

Using the Configuration Wizard Scripts to Start a Managed Server

When you use the Configuration Wizard to create a domain, the wizard creates a script that you can use to start a Managed Server:

- `domain_name\startManagedWebLogic.cmd` (Windows)
- `domain_name/startManagedWebLogic.sh` (UNIX and Windows. On Windows, this script supports the MKS and Cygnus BASH UNIX shell emulators.)

  where *`domain_name`* is the directory in which you located your domain.

Similar to the script for starting an Administration Server, `startManagedWebLogic` script sets values for some domain-specific variables. However, `startManagedWebLogic` also specifies the listen address of the domain's Administration Server, which causes the server to run as a Managed Server and retrieve its configuration from the Administration Server.

Before you use `startManagedWebLogic`, open the script in a text editor and make sure that the `SERVER_NAME` variable is set to the name of the WebLogic Managed Server that you wish to start. Also verify that the `ADMIN_URL` specifies the host (host name or IP address) and port number where the Administration Server is listening for requests (default is 7001). For example:

```
set SERVER_NAME=bigguy
set ADMIN_URL=peach:7001
```

Instead of opening and modifying `startManagedWebLogic`, you can enter either of the following commands:

- `domain_name\startWebLogic managed_server_name admin_url`

  By passing two parameters to the script that starts an Administration Server, you can start a Managed Server.

- `domain_name\startManagedWebLogic managed_server_name admin_url`

  The above syntax overrides the values of the `SERVER_NAME` and `ADMIN_URL` in the `startManagedWebLogic` script.

For example, the following command uses `startWebLogic.cmd` to start a managed server named myManagedServer using the Administration Server named peach that listens on port 7001:

```
c:\user_domains\mydomain\startWebLogic.cmd myManagedServer http://peach:7001
```

For a complete description of the variables and Java options that can be specified in `startManagedWebLogic`, see Table 2-1 under "Starting an Administration Server Using a Script" on page 2-14.

For more information on configuring a connection to the Administration Server, refer to "Configuring a Connection to the Administration Server" on page 2-38.

When the server has successfully completed its startup process, it writes the following message to the command window:

```
<Notice> <WebLogicServer> <000360> <Server started in RUNNING mode>
```

Creating Your Own Script to Start a Managed Server

If you use some other means to create a domain (such as the Administration Console), you can create your own startup script that starts a Managed Server in your domain. The steps for creating such a script are the same as the steps described in "Creating Your Own Script to Start an Administration Server" on page 2-14 with the following addition:

You must set a value for a variable named `ADMIN_URL`. For information on configuring a connection to the Administration Server, refer to "Configuring a Connection to the Administration Server" on page 2-38.

When the server has successfully completed its startup process, it writes the following message to the command window:

```
<Notice> <WebLogicServer> <000360> <Server started in RUNNING mode>
```

Starting a Managed Server from the Command Line

To start a WebLogic Managed Server from a command line, you use same command and arguments that you use for an Administration Server **plus** one of the following arguments, which configures a connection to the Administration Server:

- `-Dweblogic.management.server=`*`host:port`*
- `-Dweblogic.management.server=http://`*`host:port`*
- `-Dweblogic.management.server=https://`*`host:port`*

For information on configuring a connection to the Administration Server, refer to "Configuring a Connection to the Administration Server" on page 2-38.

For information on the command and arguments for starting an Administration Server, refer to "Using the weblogic.Server Command" on page 2-17.

When the server has successfully completed its startup process, it writes the following message to the command window:

```
<Notice> <WebLogicServer> <000360> <Server started in RUNNING mode>
```

Configuring a Connection to the Administration Server

Regardless of whether you start a Managed Server from the Windows Start menu, a script, or the `weblogic.Server` command, you must make sure that the Managed Server specifies the correct listen address of the Administration Server. A Managed Server uses this address to retrieve its configuration from the Administration Server.

Note: The first time you start a Managed Server, it must be able to contact the Administration Server. Thereafter you can configure Managed Servers to start even if the Administration Server is unavailable. For more information, refer to Starting a Managed Server When the Administration Server Is Not Available in the *Creating and Configuring WebLogic Server Domains* Guide.

You can express the listen address in one of the following formats:

- *`host:port`*

  where *`host`* is the name or IP address of the machine where the Administration Server is running and *`port`* is the Administration Server's default, non-SSL listen port. (By default the Administration Server's listen port is 7001.)

  With this format, the Managed Server uses its default protocol (`t3`) to access the Administration Server. To modify the default protocol, do the following:

  a. Start the Administration Server.

  b. From the Administration Console, in the left pane, expand the Servers node and click the name of the Managed Server.

  c. In the right pane, click Connections→Protocols.

  d. The Default Protocol field determines the default protocol for a server.

- `http://`*`host`*`:`*`port`*

  where *`host`* is the name or IP address of the machine where the Administration Server is running and *`port`* is the Administration Server's default, non-SSL listen port. (By default the Administration Server's listen port is 7001.)

  To verify the host IP address, name, and default listen port of the Administration Server, start the Administration Server in a command shell. When the server successfully finishes its startup cycle, it prints to standard out messages that are similar to the following (among other messages):

  ```
  <Apr 19, 2002 9:24:19 AM EDT> <Notice> <WebLogicServer>
  <000355> <Thread "Listen Thread.Default" listening on port
  7001, ip address 11.12.13.141>
  ```

  ...

  ```
  <Apr 19, 2002 9:24:19 AM EDT> <Notice> <WebLogicServer>
  <000331> <Started WebLogic Admin Server "myserver" for domain
  "mydomain" running in Development Mode>
  ```

  You can change the IP address and listen port values from the Administration Console on a server's Configuration→General tab.

- `https://`*`host:port`*

  If you have configured Secure Socket Layer (SSL) communication for the Managed Server and Administration Server, you can use this format. In this format, *`host`* is the name or IP address of the machine where the Administration Server is running and *`port`* is the Administration Server's SSL listen port.

  If you set up the Administration Server to use an Administration Port, *`port`* must specify the Administration Port.

  For information on enabling SSL, refer to Configuring the SSL Protocol in the Administration Console Online Help. For more information on Administration Ports, refer to Configuring a Domain-Wide Administration Port in the *Creating and Configuring WebLogic Server Domains* Guide.

Because the Managed Server receives its configuration from the Administration Server, the Administration Server specified must be in the same domain as the Managed Server.

Specifying the Default Startup State

To set up a server so that the `weblogic.Server` command (or a script that executes the command) starts it in `STANDBY` by default, do the following (starting a server in `STANDBY` requires you to set up an Administration Port for the server):

1. In the Administration Console, expand the Servers node in the left pane. A list of servers appears under the Servers node.
2. Select a specific server in the left pane.
3. On the General tab, in the Startup Mode field, enter `STANDBY`.
4. Click Apply to save your changes.

Starting a Remote Managed Server

If a Node Manager is running on the host machine of a Managed Server, you can start the Managed Server from a remote host using the Administration Console or the `weblogic.Admin` utility. **Node Manager** is a standalone Java program provided with WebLogic Server that enables you to start and stop remote Managed Servers.

For information about starting a remote server from the Administration Console, refer to Starting a Server and Starting a Server in the STANDBY State in the Administration Console Online Help.

For information on using the `weblogic.Admin` command utility, refer to "START" on page B-26 and "STARTINSTANDBY" on page B-28.

For information about the Node Manager, refer to Managing Server Availability with Node Manager in the *Creating and Configuring WebLogic Server Domains* Guide.

Starting and Killing All WebLogic Servers in a Domain or Cluster

If the Node Manager is running on the host machines of your Managed Servers, you can use the Administration Console to start all of the Managed Servers in the domain or in a specific cluster. You cannot start the Administration Server from the Administration Console.

You can also use the Administration Console to force a shutdown (kill) of all WebLogic Servers in a domain or in a cluster. The kill command initiates a forced shutdown for Managed Servers and the Administration Server. It does not require the Node Manager.

This section describes the following tasks:

- Starting All Managed Servers in a Domain
- Starting All Managed Servers in a Cluster
- Killing All Servers in a Domain
- Killing All Servers in a Cluster

For information about the Node Manager, refer to Managing Server Availability with Node Manager in the *Creating and Configuring WebLogic Server Domains* Guide.

Starting All Managed Servers in a Domain

To start all of the Managed Servers in the active domain, do the following:

1. Start the Administration Server for the domain.
2. Start the Node Manager on all machines in the domain. For more information, refer to Starting Node Manager in the *Creating and Configuring WebLogic Server Domains* Guide.
3. From the Administration Console, right click on the name of the active domain in the left panel.
4. Select **Start this domain...**

5. When the Administration Console prompts you to confirm the command, click Yes.

   The Administration Console displays a page that lists the name of each WebLogic Server in the domain.

6. To view the result of the start operation for a server, click its name.

Starting All Managed Servers in a Cluster

To start all of the Managed Servers in a cluster, do the following:

1. Start the Administration Server for the domain.
2. Start the Node Manager on all machines in the cluster. For more information, refer to Starting Node Manager in the *Creating and Configuring WebLogic Server Domains* Guide.
3. From the Administration Console, right click on the name of the cluster in the left panel.
4. Select **Start this cluster...**
5. When the Administration Console prompts you to confirm the command, click Yes.

   The Administration Console displays the Tasks page, which displays the status of the startup task for each Managed Server in the cluster.

6. To view details about a server's startup status, on the Tasks page, click the startup task's name. Then click the Details tab.

Killing All Servers in a Domain

To initiate a force shutdown (kill) for all servers in a domain, do the following:

1. From the Administration Console, right click on the name of the cluster in the left panel.
2. **Kill this domain...**
3. When the Administration Console prompts you to confirm the command, click Yes.

Managed Servers and the Administration Server immediately stop all work items and shut down. If a Managed Server does not respond, and if you used the Node Manager to start the server, the Node Manager kills the server.

4. To confirm that the domain is killed, review the output in the shell process that runs the Administration Server. It displays an `ALERT` message that indicates the shutdown sequence has been initiated, and then it exits the process.

Killing All Servers in a Cluster

To initiate a force shutdown (kill) for servers in a cluster, do the following:

1. From the Administration Console, right click on the name of the cluster in the left panel.
2. **Kill this domain...**
3. When the Administration Console prompts you to confirm the command, click Yes.

   All servers in the cluster immediately stop all work items and shut down. If a Managed Server does not respond, and if you used the Node Manager to start the server, the Node Manager kills the server.
4. To confirm that the cluster is killed, do one of the following:
   - If the Administration Server is not part of the cluster, in the left pane, click the Tasks node. On the Tasks page, click the shutdown task's name. Then click the Details tab.
   - If the Administration Server is part of the cluster, review the output in the shell process that runs the Administration Server. It displays an `ALERT` message that indicates the shutdown sequence has been initiated, and then it exits the process.

Shutting Down WebLogic Servers

You can do any of the following to shut down a WebLogic Server:

- Using the Administration Console:

    - Shutting Down a Server
    - Forcing Shutdown of a Server

- Using the `weblogic.Admin` utility:
    - "SHUTDOWN" on page B-24
    - "FORCESHUTDOWN" on page B-12

When you initiate a graceful shutdown, the server notifies subsystems to complete all in-work requests. After the subsystems complete their work, the server stops.

When you initiate a forced shutdown, the server instructs subsystems to immediately drop in-work requests. If you force a Managed Server to shut down and it fails to respond, and if you started the server with the Node Manager, the Node Manager kills the server process.

The server waits a number of seconds for all subsystems to successfully stop. After the number of seconds expires, the server does one of the following:

- If the timeout occurs when the server is in the `RUNNING` state, the server returns a message to standard out. To shut down the server after this occurs, you must issue a force shutdown command.
- If the timeout occurs when the server is in the `STANDBY` or `SHUTTING_DOWN` state, it kills all processes and shuts down.

For information on changing the default number of seconds, refer to Setting the Timeout Period for LifeCycle Operations in the Administration Console Online Help.

Configuring Startup and Shutdown Classes

You can use startup and shutdown classes to configure a WebLogic Server to perform tasks when you start or gracefully shut down the server. A startup class is a Java program that is automatically loaded and executed when a WebLogic Server is started or restarted.

By default, startup classes are loaded and executed after all other server subsystems have initialized and after the server deploys modules. For any startup class, you can override the default and specify that the server loads and executes it before the server deploys JDBC connection pools, Web applications, and EJBs.

A shutdown class is a Java program that is automatically loaded and executed when the WebLogic Server is shut down either from the Administration Console or the `weblogic.admin shutdown` command. For more information about when a server invokes startup and shutdown classes, refer to "The Server Lifecycle" on page 2-1.

To use startup or shutdown classes, you must configure and assign these classes to one or more servers. To configure a class from the Administration Console, refer to Startup and Shutdown Classes in the Administration Console Online Help. BEA provides the `examples.jms.startup` API code example which demonstrates how to establish a JMS message consumer from a WebLogic startup class.

Note: WebLogic Server 7.0 optionally installs API code examples in *`WL_HOME`*\samples\server\src\examples, where *`WL_HOME`* is the top-level directory of your WebLogic Server installation. You can start the examples server, and obtain information about the samples and how to run them.

Setting Up a WebLogic Server Instance as a Windows Service

If you want a WebLogic Server instance to start automatically when you boot a Windows host computer, you can set up the server as a Windows service.

For each server instance that you set up as a Windows service, WebLogic Server creates a key in the Windows Registry under `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services`. The registry entry contains such information as the name of the server and other startup arguments.

When you start the Windows host, the Windows Service Control Manager (SCM), which is part of the Windows operating system, uses the information in the Windows Registry key to invoke the `weblogic.Server` main class. The Windows SCM cannot be configured to use a Node Manager to start Managed Servers, and therefore the Node Manager's monitoring and automatic restart features cannot be used for servers that run as a Windows service.

The following tasks set up and manage WebLogic Server instances that run as Windows Services:

- Setting Up a Windows Service: Main Steps
- Verifying the Setup
- Using the Control Panel to Stop or Restart a Server Instance
- Removing a Server as a Windows Service
- Changing Startup Credentials for a Server Set Up as a Windows Service

Setting Up a Windows Service: Main Steps

To set up a Windows service:

1. Do **one** of the following:
   - If the Domain Configuration Wizard prompts you to install a server as a Windows Service, choose Yes.

     Some of the domain templates in the Domain Configuration Wizard prompt you to set up a server as a Windows service. If you choose Yes, the wizard installs the server as a Windows service. If you are using the wizard to create an Administration Server and Managed Servers, the wizard creates a Windows service only for the Administration Server. The wizard also creates a server-specific script for you that you can modify and use to install other servers as Windows services. The script is named `*domain-name*\installService.cmd`, where `*domain-name*` is the name of the domain that you created.
   - Create a script that sets values for server-specific variables and then calls a WebLogic Server master script. For more information, refer to "Create a Server-Specific Script" on page 2-47.
2. If you are installing a Managed Server as a Windows service, you must set additional variables in the server-specific script. For more information, refer to "Set Additional Values for Managed Servers" on page 2-50.

3. If you set up both an Administration Server and a Managed Server to run as Windows services on the same computer, modify the master script so that the Managed Server starts only after the Administration Server finishes its startup cycle. For more information, refer to "Require Managed Servers to Start After Administration Servers" on page 2-51.

4. If you want a server instance to shut down gracefully when you use the Windows Control Panel to stop the Windows service, create a Java class and modify the master script so that the Windows SCM will invoke the class. For more information, refer to "Enable Graceful Shutdowns from the Control Panel" on page 2-53.

5. If you want to see the messages that a server instance prints to standard out and standard error (including stack traces and thread dumps), redirect standard out and standard error to a file. For more information, refer to "Redirecting Standard Out and Standard Error to a File" on page 2-56.

6. If you have created additional Java classes that you want the WebLogic Server instance to invoke, you must add them to the server's classpath. For more information, refer to Adding Classes to the Classpath.

7. Run the server-specific script. For more information, refer to "Run the Server-Specific Script" on page 2-59.

Create a Server-Specific Script

If the Domain Configuration Wizard did not already create a server-specific script for your domain, you can create one. The script sets values for variables that identify the name of the server instance and other server-specific information. Then the script calls a master script, `WL_HOME\server\bin\installSvc.cmd`, where `WL_HOME` is the directory in which you installed WebLogic Server. The master scripts invokes the `beasvc` utility, which adds a key to the Windows Registry.

Note: For more information about `beasvc`, enter the following command at a command prompt: *`WL_HOME`*`\server\bin\beasvc -help`, where *`WL_HOME`* is the directory in which you installed WebLogic Server.

To see an example of a server-specific script, refer to Listing 2-1, "Example Script for Setting Up a Server as a Windows Service," on page 2-50.

To create a server-specific script:

1. In the root directory for the domain's Administration Server (the directory that contains the domain's `config.xml` file), create a text file.
2. Add the following, **required** `batch` commands to the text file, each command on a separate line:
   - `SETLOCAL`

     This is a `batch` command that begins the localization of environment variables in a batch file.
   - `set DOMAIN_NAME=`*`domain-name`*

     where *`domain-name`* is the name of your WebLogic Server domain.
   - `set USERDOMAIN_HOME=`*`absolute-pathname`*

     where *`absolute-pathname`* is the absolute pathname of the Administration Server's root directory (the directory that contains the domain's configuration file). For more information about the root directories for servers, refer to "A Server's Root Directory" on page 2-30.
   - `set SERVER_NAME=`*`server-name`*

     where *`server-name`* is the name of an existing server instance that you want set up as a Windows service.
3. Add the following **optional** `batch` commands to the text file. Place each command on a separate line:
   - `set WLS_USER=`*`username`*
     `set WLS_PW=`*`password`*

     where *`username`* is the name of an existing user with privileges to start a server instance and *`password`* is the user's password. The `beasvc` utility encrypts the login credentials and stores them in the Windows registry.

     This is one of two possible methods for avoiding the username/password prompt when a server instance starts. The disadvantage to this method is that changing the username or password for the server instance requires you to delete the Windows service and set up a new one with the new username and password. Instead of this method, you can use a boot identity file. With a boot identity file, you can change the login credentials without needing to modify the Windows service. For more information, refer to "Bypassing the Prompt for Username and Password" on page 2-7.
   - `set STARTMODE=[`*`true`*`]`

When the `STARTMODE` variable is set to `true`, the server instance starts in production mode. When not specified, or when set to `false`, the server starts in development mode. For more information about development mode and production mode, refer to "Development Mode vs. Production Mode" on page 2-28.

- `set JAVA_OPTIONS=`*`java-options`*

  where *`java-options`* is one or more Java arguments that you want to pass to the Java Virtual Machine (JVM). Separate multiple arguments with a space. For a list of Java options that are specific to WebLogic Server, refer to "Command Syntax for weblogic.Server" on page 2-19. The JVM that you use supports additional options and are documented by the JVM vendor.

- `set JAVA_VM=-`*`JVM-mode`*

  where *`JVM-mode`* is a text string that indicates the mode in which you want the JVM to run. The values that you supply depend on the JVM that you are using. If you use the JRockit JVM, the default value is `-jrockit`. For more information, refer to "Starting and Configuring the JRockit JVM" in the JRockit *User Guide*.

  To see a list of supported JVMs, refer to the List of Supported Platforms at the following URL:
  http://e-docs.bea.com/wls/certifications/certs_700/overview.html.

- `set MEM_ARGS=[-Xms`*`Number`*`m] [-Xmx`*`Number`*`m]`

  where *`Number`* is a numerical value in megabytes (MB). The`-Xms`*`Number`*`m` argument establishes a minimum heap size for the JVM and the `-Xmx`*`Number`*`m` sets a maximum heap size. By default, the minimum heap size is 23 MB and the maximum heap size is 200 MB.

4. Add the following **required** commands to the end of the script:

   - `call "`*`WL_HOME`*`\server\bin\installSvc.cmd"`

     where *`WL_HOME`* is an absolute pathname for the directory in which you installed WebLogic Server. This command calls the WebLogic Server master script.

   - `ENDLOCAL`

     This is a `batch` command that ends the localization of environment variables in a batch file.

5. Save the text file with a `.cmd` extension. By default, the Windows command prompt associates the `.cmd` extension with `batch` files.

Set Additional Values for Managed Servers

If you want to install a Managed Server as a Windows service, you must include a variable that specifies the location of the domain's Administration Server. The Managed Server must contact the Administration Server to retrieve its configuration data.

To set additional values for Managed Servers:

1. In a text editor, open the server-specific script.

   If you are modifying a script that the Domain Configuration Wizard created, BEA recommends that you make a copy of *`domain-name`*`\installService.cmd` (where *`domain-name`* is the name of the domain that you created) and open the copy.

2. In the text file, between the `SETLOCAL` command and the `call` command, create the following command:

   ```
   set ADMIN_URL=protocol://listen-address:listen-port
   ```

   where

   - *`protocol`* is `http` or `https`
   - *`listen-address`* is a listen address of the Administration Server
   - *`listen-port`* is a port of the Administration Server

   For more information, refer to "Configuring a Connection to the Administration Server" on page 2-38.

   For an example, refer to the bold text in Listing 2-1.

3. Save your modifications to the server-specific script.

Listing 2-1 Example Script for Setting Up a Server as a Windows Service

```
echo off
SETLOCAL
```

```
set DOMAIN_NAME=myWLSdomain
set USERDOMAIN_HOME=d:\bea\user_projects\myWLSdomain
set SERVER_NAME=myWLSserver
set STARTMODE=true
set JAVA_OPTIONS=-Dweblogic.Stdout="d:\bea\user_projects\myWLSdomain\stdout.txt"
-Dweblogic.Stderr="d:\bea\user_projects\myWLSdomain\stderr.txt"

set ADMIN_URL=http://adminserver:7501

set MEM_ARGS=-Xms40m -Xmx250m

call "d:\bea\weblogic700\server\bin\installSvc.cmd"

ENDLOCAL
```

Require Managed Servers to Start After Administration Servers

If you set up both an Administration Server and a Managed Server to run as Windows services on the same computer, you can specify that the Managed Server starts only after the Administration Server.

To require a Managed Server to start after the Administration Server Windows service, do the following:

1. Create a backup copy of the `WL_HOME\server\bin\installSvc.cmd` master script.
2. If you have already installed the Administration Server as a Windows Service, remove the service. For more information, refer to "Removing a Server as a Windows Service" on page 2-62.
3. Before you install (or reinstall) the Administration Server as a Windows Service, do the following:
   a. In a text editor, open the `WL_HOME\server\bin\installSvc.cmd` master script.

      The last command in this script invokes `beasvc`, which is the WebLogic Server utility that modifies the Windows Registry.
   b. In `installSvc.cmd`, add the following argument to the command that invokes the `beasvc` utility:

      `-delay:`*`delay_milliseconds`*

This specifies the number of milliseconds to wait before the Windows SCM changes the service status from `SERVER_START_PENDING` to `STARTED`.

For example, if your Administration Server requires 2 minutes to complete its startup cycle and begin listening for requests, then specify `-delay=120000`. When you boot the Windows host computer, the Windows SCM reports a status of `SERVER_START_PENDING` for 2 minutes. Then it changes the status to `STARTED`.

The modified `beasvc` invocation for the Administration Server will resemble the following:

```
"%WL_HOME%\server\bin\beasvc" -install
-svcname:"%DOMAIN_NAME%_%SERVER_NAME%"
-delay:120000
-javahome:"%JAVA_HOME%" -execdir:"%USERDOMAIN_HOME%"
-extrapath:"%WL_HOME%\server\bin" -password:"%WLS_PW%"
-cmdline:%CMDLINE%
```

For more information about `beasvc`, enter the following command at a command prompt: *`WL_HOME`*`\server\bin\beasvc -help`, where *`WL_HOME`* is the directory in which you installed WebLogic Server.

4. Install the Administration Server Windows service.
5. Before you install the **Managed Server** as a Windows service, do the following:
   a. In a text editor, open the `WL_HOME\server\bin\installSvc.cmd` master script.
   b. In `installSvc.cmd`, add the following argument to the command that invokes the `beasvc` utility:

      `-depend:`*`Administration-Server-service-name`*

      where *`Administration-Server-service-name`* is the name of the Administration Server Windows service. To verify the service name, look on the Windows Services Control Panel.

      With this option, the Windows SCM will wait for the Administration Server Windows service to report a status of `STARTED` before it starts the Managed Server Windows service.

      For example, the modified `beasvc` invocation for the Managed Server will resemble the following:

```
"%WL_HOME%\server\bin\beasvc" -install
-svcname:"%DOMAIN_NAME%_%SERVER_NAME%"
-depend:"myDomain_myAdminServer"
-javahome:"%JAVA_HOME%" -execdir:"%USERDOMAIN_HOME%"
-extrapath:"%WL_HOME%\server\bin" -password:"%WLS_PW%"
-cmdline:%CMDLINE%
```

You can also add the `-delay:`*`delay_milliseconds`* option to a Managed Server Windows service if you want to configure when the Windows SCM reports a status of `STARTED` for the service.

Enable Graceful Shutdowns from the Control Panel

By default, if you use the Windows Control Panel to stop a server instance, the Windows Service Control Manager (SCM) kills the server's Java Virtual Machine (JVM). If you kill the JVM, the server immediately stops all processing. Any session data is lost. If you kill the JVM for an Administration Server while the server is writing to the `config.xml` file, you can corrupt the `config.xml` file.

To enable graceful shutdowns from the Windows Control Panel, do the following:

1. Create a Java class that invokes the `weblogic.management.runtime.ServerRuntime.shutdown()` method.

   This method gracefully shuts down a server after the server has completed all inflight work. For an example of such a class, refer to "Java Class that Shuts Down a Server Instance" on page 2-55.

2. Create a backup copy of the `WL_HOME\server\bin\installSvc.cmd` master script.

3. In a text editor, open the `WL_HOME\server\bin\installSvc.cmd` master script and do the following:

   a. Add the class that you created to the `set CLASSPATH` statement.

      For example if you archived your class in a file named `c:\myJar`, the modified statement will be as follows:

      ```
      set
      CLASSPATH=%JAVA_HOME%\lib\tools.jar;%WL_HOME%\server\lib\web
      logic_sp.jar;%WL_HOME%\server\lib\weblogic.jar;c:\myJar;%CLA
      SSPATH%
      ```

b. Add the following argument to the last line of the script, which calls the `beasvc` utility:
`-stopclass:`*`javaclass`*
where *`javaclass`* is the full classpath name of the class that you created. This argument loads *`javaclass`* and then invokes its `public void static stop()` method.

For example, if you packaged the class in "Java Class that Shuts Down a Server Instance" on page 2-55 in `com.myClasses`, the modified `beasvc` command will be as follows:
the modified `beasvc` invocation will resemble the following:

```
"%WL_HOME%\server\bin\beasvc" -install
-svcname:"%DOMAIN_NAME%_%SERVER_NAME%"
-stopclass:com.myClasses.ServerStopper
-javahome:"%JAVA_HOME%" -execdir:"%USERDOMAIN_HOME%"
-extrapath:"%WL_HOME%\server\bin" -password:"%WLS_PW%"
-cmdline:%CMDLINE%
```

For more information about `beasvc`, enter the following command at a command prompt: *`WL_HOME`*`\server\bin\beasvc -help`, where *`WL_HOME`* is the directory in which you installed WebLogic Server.

4. Consider modifying the default timeout value that the Windows SCM specifies.

   By default, when you use the Windows 2000 Control Panel to stop a Windows service, the Windows SCM waits 30 seconds for the service to stop before it kills the service and prints a timeout message to the System event log.

   If you use `-stopclass` to gracefully shut down a server, 30 seconds might not be enough time for the server to gracefully end its processing.

   To configure a timeout period on Windows 2000, create a `REG_DWORD` registry value named `ServicesPipeTimeout` under the following registry key:
   `HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control`

   The key value must be in milliseconds.

   This value is read from the registry during the startup of the Windows operating system and it affects all services that are installed.

5. Save your changes to the WebLogic Server master script.

Java Class that Shuts Down a Server Instance

The following Java class uses Java Management Extensions (JMX) to shut down a server instance. Each server uses JMX Managed Beans (MBeans) to expose its management attributes and operations. One such MBean, `ServerRuntime`, exposes a `shutdown()` method that gracefully shuts down a server.

The class in Listing 2-2 uses the Administration `MBeanHome` interface, which can retrieve and call `ServerRuntime` MBean operations for all server instances in a domain.

For more information about JMX programming, refer to the Programming WebLogic Management Services with JMX guide.

Listing 2-2 Java Class that Shuts Down a Server Instance

```
import java.util.Set;
import java.util.Iterator;
import java.rmi.RemoteException;
import javax.naming.*;
import weblogic.jndi.Environment;

import weblogic.management.MBeanHome;
import javax.management.ObjectName;
import weblogic.management.WebLogicMBean;
import weblogic.management.configuration.ServerMBean;
import weblogic.management.runtime.ServerRuntimeMBean;
import weblogic.management.WebLogicObjectName;

public class ServerStopper {
  public static void stop() throws Exception {
    MBeanHome home = null;

    //url of the Admin server
    String url = "t3://qa113:7001";
    String username = "system";
    String password = "gumby1234";
    ServerRuntimeMBean serverRuntime = null;
    Set mbeanSet = null;
    Iterator mbeanIterator = null;

    try {
      // Set ContextClassloader to prevent assertions
      URL[] urls = { new File("/").toURL() };
      Thread.currentThread().setContextClassLoader(new
         URLClassLoader(urls));
```

```
            Environment env = new Environment();
            env.setProviderUrl(url);
            env.setSecurityPrincipal(username);
            env.setSecurityCredentials(password);
            Context ctx = env.getInitialContext();
            home = (MBeanHome)
                 ctx.lookup("weblogic.management.adminhome");
            mbeanSet = home.getMBeansByType("ServerRuntime");
            mbeanIterator = mbeanSet.iterator();

            while(mbeanIterator.hasNext()) {
              serverRuntime = (ServerRuntimeMBean)mbeanIterator.next();
              if(serverRuntime.getState().equals("RUNNING")) {
                serverRuntime.shutdown();
              }
            }

          } catch (Exception e) {
            e.printStackTrace();
          }
        }
      }
```

Redirecting Standard Out and Standard Error to a File

By default, when you install a WebLogic Server instance as a Windows service, you cannot see the messages that the server or its JVM print to standard out and standard error.

To view these messages, you must redirect standard out and standard error to a file:

1. Create a backup copy of the `WL_HOME\server\bin\installSvc.cmd` master script.
2. In a text editor, open the `WL_HOME\server\bin\installSvc.cmd` master script.
3. In `installSvc.cmd`, the last command in the script invokes the `beasvc` utility. At the end of the `beasvc` command, append the following command option:

   `-log:"`*`pathname`*`"`
   where *`pathname`* is a fully qualified path and filename of the file that you want to store the server's standard out and standard error messages.

   The modified `beasvc` command will resemble the following command:

```
"%WL_HOME%\server\bin\beasvc" -install
-svcname:"%DOMAIN_NAME%_%SERVER_NAME%"
-javahome:"%JAVA_HOME%" -execdir:"%USERDOMAIN_HOME%"
-extrapath:"%WL_HOME%\server\bin" -password:"%WLS_PW%"
-cmdline:%CMDLINE%
-log:"d:\bea\user_projects\myWLSdomain\myWLSserver-stdout.txt
```

4. By default, every 24 hours the Windows service archives messages to a file named `pathname-yyyy_mm_dd-hh_mm_ss`. New messages collect in the file that you specified in the previous step.

After you install the service and restart the Windows host, to view the messages that the server writes to standard out or standard error, do one of the following:

- Make a copy of the file that you specified and view the copy. The Windows file system cannot write to files that are currently opened.
- To view the messages as they are being printed to the file, open a command prompt and use the DOS command `tail -f stdout-filename`.

Printing Thread Dumps to Standard Out

To cause the WebLogic Server instance to print a thread dump to standard out, do either of the following:

- Use the `weblogic.Admin THREAD_DUMP` command. For more information, refer to "THREAD_DUMP" on page B-36.
- Open a command prompt and enter the following command:

  ```
  WL_HOME\bin\beasvc -dump -svcname:service-name
  ```
  where `WL_HOME` is the directory in which you installed WebLogic Server and *`service-name`* is the Windows service that is running a server instance.

For example:

```
D:\bea\weblogic70\server\bin\beasvc -dump
-svcname:mydomain_myserver
```

Adding Classes to the Classpath

The **classpath** is a declaration of the location of Java classes that a JVM can invoke. When you use the WebLogic Server master script to install a server instance as a Windows service, the master script specifies all classes required to run a server instance. If you want to extend WebLogic Server by adding your own Java classes, you must add them to the classpath.

To add classes to the classpath:

1. Create a backup copy of the `WL_HOME\server\bin\installSvc.cmd` master script.
2. In a text editor, open the `WL_HOME\server\bin\installSvc.cmd` master script.
3. Add your class to the `set CLASSPATH` statement.

   For example if you archived your class in a file named `c:\myJar`, the modified statement will be as follows:

   ```
   set
   CLASSPATH=%JAVA_HOME%\lib\tools.jar;%WL_HOME%\server\lib\weblog
   ic_sp.jar;%WL_HOME%\server\lib\weblogic.jar;c:\myJar;%CLASSPATH
   %
   ```

   Note: Win32 systems have a 2K limitation on the length of the command line. If the classpath setting for the Windows service startup is very long, the 2K limitation could be exceeded.

   To work around this limitation:

   a. Place the value of the `set CLASSPATH` command in a separate text file and save the text file in the `WL_HOME\server\bin` directory.

   b. In the `WL_HOME\server\bin\installSvc.cmd` master script, find the `set CMDLINE` command.

   c. Within the `set CMDLINE` command, replace the `-classpath \"%CLASSPATH%\"` option with the following option:

   `-classpath @`*`filename`*
   where *`filename`* is the name of the file that contains the classpath values.

   For example:

   ```
   set CMDLINE="%JAVA_VM% %MEM_ARGS% %JAVA_OPTIONS%
   -classpath @myClasspath.txt -Dweblogic.Name=%SERVER_NAME%
   -Dbea.home=\"D:\bea_70sp2\"
   ```

```
-Dweblogic.management.username=%WLS_USER%
-Dweblogic.management.server=\"%ADMIN_URL%\"
-Dweblogic.ProductionModeEnabled=%STARTMODE%
-Djava.security.policy=\"%WL_HOME%\server\lib\weblogic.polic
y\" weblogic.Server"
```

4. Save your changes to the WebLogic Server master script.

Run the Server-Specific Script

Note: To run the server-specific script, you must log in to the Windows computer with a user account that has privileges to modify the Windows registry.

If you install the Windows service in a production environment, BEA recommends that you do **not** run the service under an operating-system user account that has administrator-level privileges. For more information, see "Verifying the User Account Under Which the Service Runs" on page 2-60.

To run the server-specific script:

1. Open a command prompt and change to Administration Server's root directory, which is the directory that contains the server-specific script.

2. Enter the name of the server-specific script.

   The command prompt runs the script as a `batch` file.

   If the script runs successfully, it creates a Windows service named *`DOMAIN_NAME_SERVER_NAME`* and prints a line to standard out that is similar to the following:

   ```
   mydomain_myserver installed.
   ```

   By default, standard out is the command prompt in which you run the server-specific `batch` file.

3. If you modified the `WL_HOME\server\bin\installSvc.cmd` master script, consider undoing your modifications so the script can be used to set up other server instances.

Verifying the Setup

To verify that you successfully set up a WebLogic Server as a Windows service, do the following:

1. Open a command window and enter the following command:
   `set PATH=WL_HOME\server\bin;%PATH%`
2. Navigate to the directory immediately above your domain directory. For example, to verify the setup for `BEA_HOME\user_domains\mydomain`, navigate to `BEA_HOME\user_domains`.
3. Enter:
   `beasvc -debug "yourServiceName"`

   For example, `beasvc -debug "mydomain_myserver"`.

If your setup was successful, the `beasvc -debug` command starts your server. If the script returns an error similar to the following, make sure that you specified the correct service name:

```
Unable to open Registry Key .......
System\CurrentControlSet\Services\beasvc
example_examplesServer\Parameters
```

Verifying the User Account Under Which the Service Runs

In a production environment, WebLogic Server Windows services should run under a special operating-system user account that has limited access privileges. For example, the OS user should have access privileges only to BEA files and to your domain files. This should be the only user account that has access to these files.

To ensure that the WebLogic Server instance runs under the special OS user account:

1. Open the Services control panel.

   For example, from the Windows 2000 desktop:

   a. Select the Start menu.

   b. On the Start menu, select Settings →Control Panel

   c. In the Control Panel window, open the Administrative Tools folder

   d. In the Administrative Tools window, open the Services control panel.

2. On the Services control panel, right click the WebLogic Server Windows service and click Properties.
3. In the Properties window, click the Log On tab.
4. Under Log on as, select This account. Then enter the user name and password of the special OS user account.
5. Click OK.

Using the Control Panel to Stop or Restart a Server Instance

After you set up a server instance to run as a Windows service, you can use the Service Control Panel to stop and restart the server.

By default, if you use the Windows Control Panel to stop a server instance, the Windows Service Control Manager (SCM) kills the server's Java Virtual Machine (JVM). If you kill the JVM, the server immediately stops all processing. Any session data is lost. If you kill the JVM for an Administration Server while the server is writing to the `config.xml` file, you can corrupt the `config.xml` file. For information on enabling graceful shutdowns from the Windows Control Panel, refer to "Enable Graceful Shutdowns from the Control Panel" on page 2-53.

To stop or restart a WebLogic Server instance that is installed as a Windows service:

1. Select Start→Settings→Control Panel.
2. On Windows 2000, open the Administrative Tools Control Panel. Then open the Services Control Panel.

   On Windows NT, open the Services Control Panel directly from the Control Panel window.
3. In the Services Control Panel, find the service that you created. By default, the service name starts with `beasvc`.
4. Right-click the service name and select commands from the shortcut menu.

Removing a Server as a Windows Service

To remove a Windows service that runs a WebLogic Server instance, you can use a script that causes the `beasvc` utility to remove the associated key from the Windows Registry. Removing the Windows service has no effect on the server instance's configuration that is saved in the domain's configuration file. After you remove the Windows service, you can start the WebLogic Server instance with start scripts or, for Managed Servers, the Node Manager.

If the Domain Configuration Wizard did not already create a script for your domain, you can create one. The script sets values for variables that identify the name of the server instance and other server-specific information. Then the script calls a master uninstall script, `WL_HOME\server\bin\uninstallSvc.cmd`, where `WL_HOME` is the directory in which you installed WebLogic Server. The master scripts invokes the `beasvc` utility, which removes a key from the Windows Registry.

To see an example of a server-specific uninstaller script, refer to Listing 2-3, "Script to Remove a Windows Service," on page 2-63.

To create a script for removing a Windows service that runs a WebLogic Server instance:

1. In the root directory for the domain's Administration Server (the directory that contains the domain's `config.xml` file), create a text file.
2. Add the following, **required** `batch` commands to the text file, each command on a separate line:
   - `SETLOCAL`

     This is a `batch` command that begins the localization of environment variables in a batch file.
   - `set DOMAIN_NAME=`*`domain-name`*

     where *`domain-name`* is the name of your WebLogic Server domain.
   - `set SERVER_NAME=`*`server-name`*

     where *`server-name`* is the name of an existing server instance that you want set up as a Windows service.
   - `call "`*`WL_HOME`*`\server\bin\uninstallSvc.cmd"`

where `WL_HOME` is an absolute pathname for the directory in which you installed WebLogic Server. This command calls the WebLogic Server master uninstall script.

- `ENDLOCAL`

  This is a `batch` command that ends the localization of environment variables in a batch file.

3. Save the text file with a `.cmd` extension. By default, the Windows command prompt associates the `.cmd` extension with `batch` files.
4. Enter the name of the server-specific script.

   The command prompt runs the script as a `batch` file.

If the removal script runs successfully, it prints a line similar to the following to standard out:

```
mydomain_myserver removed.
```

By default, standard out is the command prompt in which you run the `batch` file.

Listing 2-3 Script to Remove a Windows Service

```
echo off
SETLOCAL

set DOMAIN_NAME=myWLSdomain
set SERVER_NAME=myWLSserver
call "D:\bea\weblogic700\server\bin\uninstallSvc.cmd"

ENDLOCAL
```

Changing Startup Credentials for a Server Set Up as a Windows Service

To change a Windows service so that a WebLogic Server instance runs under different user credentials, do **one** of the following:

- If you set up the Windows service to retrieve usernames and passwords from a boot identity file, you can overwrite the existing file with a new one that contains the new username and password. You must specify the name of an existing user in the WebLogic Server default security realm. For information, refer to "Creating a Boot Identity File for an Administration Server" on page 2-8.
- If you set up the Windows service to retrieve usernames and passwords from the Windows registry, then you must remove the Windows service and create a new one that uses your new username or password:

1. Uninstall the Windows service that runs the WebLogic Server instance. For more information, refer to "Removing a Server as a Windows Service" on page 2-62.
2. In a text editor, open the script that you used to install the service and enter the new username and password as the value for the `set WLS_USER` and `set WLS_PW` commands. WebLogic encrypts these values in the Windows Registry.
3. Save your modifications to the script.
4. Enter the name of the server-specific script.

   The command prompt runs the script as a `batch` file.

   If the script runs successfully, it creates a Windows service named *`DOMAIN_NAME_SERVER_NAME`* and prints a line to standard out that is similar to the following:

   ```
   mydomain_myserver installed.
   ```

   By default, standard out is the command prompt in which you run the server-specific `batch` file.
5. (Optional) Remove the username and password from the script file.

3 Protecting System Administration Operations

To leverage individual skills, many Web development teams divide system administration responsibilities into distinct roles. Each project might give only one or two team members permission to deploy components, but allow all team members to view the WebLogic Server configuration. WebLogic Server supports this role-based development by providing global roles that determine access privileges for system administration operations: Anonymous, Admin, Deployer, Operator, and Monitor.

The following sections describe security roles and system administration operations:

- "Operations Available to Each Role" on page 3-2
- "Protected User Interfaces" on page 3-3
- "Permissions for Starting and Shutting Down Servers" on page 3-10

Note: These role-based permissions replace access control lists (ACLs) for securing WebLogic Server MBeans, which were used before Release 7.0.

Operations Available to Each Role

Table 3-1 describes the four global roles that WebLogic Server uses to determine access privileges for system administration operations, and the permissions granted to each role.

Table 3-1 Global Roles and Permissions

Global Role	Permissions
Anonymous	All users (the group everyone) are granted this global role. **Note:** This global role is provided as a convenience, and can be specified in the `weblogic.xml` and `weblogic-ejb-jar.xml` deployment descriptors.
Admin	■ View the server configuration, including the encrypted value of encrypted attributes. ■ Modify the entire server configuration. ■ Deploy enterprise applications, startup and shutdown classes, and Web Application, EJB, J2EE Connector, and Web Service modules. ■ Start, resume, and stop servers by default.
Deployer	■ View the server configuration, except for encrypted attributes. ■ Deploy enterprise applications, startup and shutdown classes, and Web Application, EJB, J2EE Connector, and Web Service modules.
Operator	■ View the server configuration, *except* for encrypted attributes. ■ Start, resume, and stop servers by default.
Monitor	■ View the server configuration, except for encrypted attributes. **Note:** This security role effectively provides read-only access to the WebLogic Server Administration Console, `weblogic.Admin` utility and MBean APIs.

No user, regardless of role membership, can view the non-encrypted version of an encrypted attribute.

Note: If you are working directly with WebLogic Server MBeans and want more detailed information about the global roles and their privileges than is shown in Table 3-1, see "Protected MBean Attributes and Operations" in *Securing WebLogic Resources*.

You can add to the default global roles by creating your own security roles (global or scoped) as described in "Ways to Create Security Roles in the Administration Console" in *Securing WebLogic Resources*.

Default Group Associations

By default, WebLogic Server grants four of the default global roles to four of the default groups. By adding a user to one of these groups, the user is automatically granted the global role. These default group associations are shown in Table 3-2.

Table 3-2 Default Group Associations

Members of This Group	Are In This Role
Administrators	Admin
Deployers	Deployer
Operators	Operator
Monitors	Monitor

For information on creating users and assigning them to groups, refer to Creating Users and Adding Users to Groups in the *Securing WebLogic Resources* guide.

Protected User Interfaces

The WebLogic Server Administration Console, the `weblogic.Admin` command, and MBean APIs are secured using the default security policies, which are based on the default global roles and default groups described in Table 3-1and Table 3-2. Therefore, to use the Administration Console, a user must belong to one of these default groups or be granted one of these global roles. Additionally, administrative operations that require interaction with MBeans are secured using the MBean protections described in "Protected MBean Attributes and Operations" in *Securing WebLogic Resources*. Therefore, interaction with the following protected public interfaces typically must satisfy both security schemes.

- *The WebLogic Server Administration Console*—The WebLogic Security Service verifies whether a particular user can access the Administration Console when the user attempts to log in. If a user attempts to invoke an operation for which they do not have access, they see an Access Denied error.

  For information about using this public interface, see the *Administration Console Online Help*.

- *The weblogic.Admin command*—The WebLogic Security Service verifies whether a particular user has permission to execute a command when the user attempts to invoke the command. If a user attempts to invoke an operation for which they do not have access, WebLogic Server throws a `weblogic.management.NoAccessRuntimeException`, which developers can explicitly catch in their programs. The server sends this exception to its log file, but you can also configure the server to send exceptions to standard out.

  For information about using this public interface, see "weblogic.Admin Command-Line Reference" in the *WebLogic Server Command Line Reference*.

  Note: The `weblogic.Admin` command is a convenience utility that abstracts the interaction with the MBean APIs (described below). Therefore, for any administrative task you can perform using the `weblogic.Admin` command, you can also choose to write yourself using the MBean APIs.

- *MBean APIs*—The WebLogic Security Service verifies whether a particular user has permission to access the API when the user attempts to perform an operation on the MBean. If a user attempts to invoke an operation for which they do not have access, WebLogic Server throws a `weblogic.management.NoAccessRuntimeException`, which developers can explicitly catch in their programs. The server sends this exception to its log file, but you can also configure the server to send exceptions to standard out.

  For information about using these APIs, see *Programming WebLogic Management Services with JMX*.

Layered Security Scheme for Server Resources

The following sections provide more information about the layered security scheme for Server resources:

- "Security Policies for Server Resources" on page 3-5
- "MBean Protections" on page 3-6
- "How the WebLogic Security Service Verifies Layered Protections" on page 3-6
- "Example" on page 3-7
- "Maintaining a Consistent Security Scheme" on page 3-9

Security Policies for Server Resources

Like other types of WebLogic resources, a Server resource is secured with security policies using the WebLogic Server Administration Console.

More specifically, all server resources inherit a default security policy that is based on the `Admin` and `Operator` default global security roles. As described in "Operations Available to Each Role" on page 3-2, the `Admin` and `Operator` global roles are given specific privileges that are required in order for administrators to interact with administrative interfaces like the Administration Console or the `weblogic.Admin` command. These default global roles are based on the default groups (described in "Default Group Associations" on page 3-3). Therefore, administrators who need access to Server resources need to be members of either the `Administrators` or `Operators` default groups.

Note: Because WebLogic Server grants the four default global roles to four default groups, adding a user to one of these groups automatically grants the user the global role.

Warning: Do not modify the default security policies for Server resources to make them more restrictive. Eliminating some of the existing security roles might negatively impact the functioning of WebLogic Server. However, if you like, you can make the default security policies more inclusive (for example, by adding new security roles).

MBean Protections

Each type of WebLogic resource (including a Server resource) exposes a set of its operations through its own implementation of the `weblogic.security.spi.Resource` interface (the `weblogic.security.service.ServerResource` class for Server resources). Therefore, the `ServerResource` class is the entity that is actually secured by the security policy described in "Security Policies for Server Resources" on page 3-5.

In WebLogic Server, the configuration of a Server resource is exposed through a set of MBeans. As such, the actions that the `ServerResource` class protects correspond to underlying MBean attributes and operations. For example, the `Resource` interface's `start()` method maps directly to the `start` operation of the `ServerRuntime` MBean.

The MBeans that expose the configuration of a Server resource are protected using one of the four default global roles. This protection is *in addition to* the security policy on the Server resource and is currently an unmodifiable part of the WebLogic Security Service. Therefore, while you can create your own global roles for securing Server resources, only users granted one of default global roles can view or change the configuration of a server.

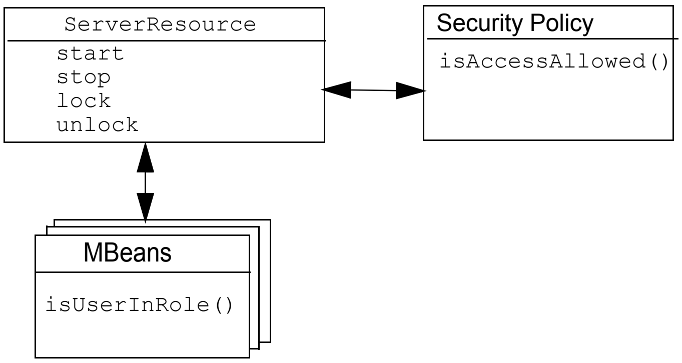
How the WebLogic Security Service Verifies Layered Protections

When an administrator tries to interact with a Server resource, the WebLogic Security Service:

- Determines whether the user is granted one of the default global roles permitted to change the attributes or invoke the operations of the MBean.
- Checks the default security policy for the Server resource to verify that the user meets the requirement defined by that security scheme.

Therefore, a user must satisfy both security schemes for their request to be successful. Figure 3-1 provides a visual representation of how a security policy on the Server resource interacts with the security role-based protections on the underlying MBeans.

Figure 3-1 Layered Protections for Server Resources



Because the privileges given by the MBean protections are immutable, it is necessary to maintain security policies in a way that ensures consistency. (For more information, see "Maintaining a Consistent Security Scheme" on page 3-9.)

Example

This example illustrates how one Server resource is protected by the layered security scheme.

An administrator with the user name `JDoe` wants to start the server called `myserver`. This administrative user (`JDoe`) is a member of the default group `Administrators`, which by default is granted the `Admin` global security role. This user-to-group and group-to-security role configuration was set up using the WebLogic Server Administration Console, as described in other sections of this guide.

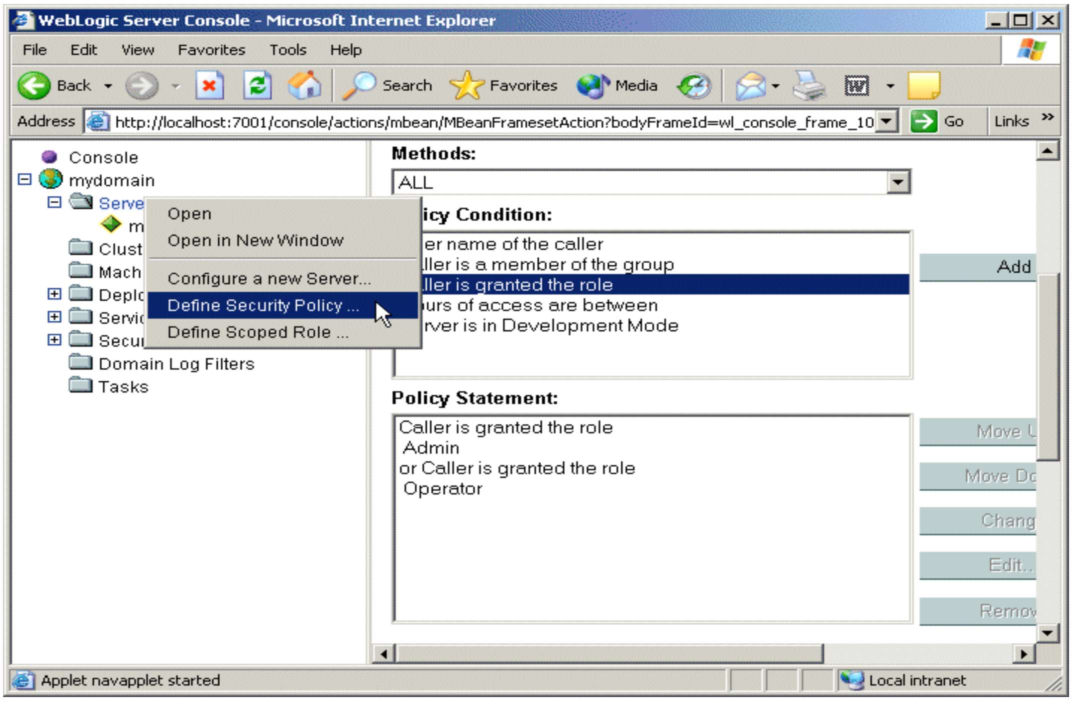
Part 1: MBean Protections

Because starting a server requires interactions with various MBeans, and because MBean protections are an immutable part of the WebLogic Security Service, a user wanting to perform such an operation must be in the `Admin` or `Operator` default global roles. For example, access to the `Server` and `ServerRuntime` MBeans (MBeans with start operations) is a privilege given only to users in these default security roles. Because the administrative user JDoe is a member of the default group Administrators, he is also granted the Admin global security role, and therefore fulfills the first part of the dual security scheme for Server resources.

Part 2: Security Policy on the Server Resource

As the Policy Statement list box in Figure 3-2 shows, the default security policy for myserver (viewed by right-clicking on myserver in the navigation tree and selecting the Define Security Policy... option) allows users granted the Admin or Operator global roles to interact with this Server resource. Because the administrative user JDoe is a member of the default group Administrators, he is also granted the Admin global security role, and therefore fulfills the second part of the layered security scheme for Server resources.

Figure 3-2 Default Security Policy for the myserver Server



Address: http://localhost:7001/console/actions/mbean/MBeanFramesetAction?bodyFrameId=wl_console_frame_10

Methods: ALL

icy Condition:
er name of the caller
ller is a member of the group
ller is granted the role
urs of access are between
rver is in Development Mode

Policy Statement:
Caller is granted the role
Admin
or Caller is granted the role
Operator

Note: Had the administrative user `JDoe` been a member of the `Operators` group (and therefore granted the `Operator` default global role), he would have still fulfilled both parts of the dual security scheme.

Maintaining a Consistent Security Scheme

WebLogic Server's default configuration of groups, global roles, security policies on Server resources, and MBean protections work together to create a consistent security scheme. You can, however, make modifications that limit access in ways that you do not intend. You must be sure that any modifications you make to the default security settings do not prevent a user from being authorized by both the MBean protections and the security policy on the Server resource.

For example, if you use the WebLogic Server Administration Console to add a user to the `Operator` global role, but fail to use the `Operator` global role in the security policy defined for a Server resource, the user can call MBean operations that are used in the startup and shutdown sequence, but cannot use any server-resource operations to start or stop a server. Similarly, if you use the Administration Console to remove the `Operator` global role from a security policy on the Server resource, a user granted the `Operator` global role can still call MBean operations but cannot call the Server resource. This result occurs because MBean protections for the default global roles are part of the WebLogic Security Service and are currently unmodifiable.

To keep MBean protections synchronized with security policies, consider taking the following actions when you create or modify a security policy:

- Always give the `Admin` global role access to a Server resource.
- For a security policy on a server, use the `Operator` global role.

  Note: Failure to use the `Operator` global role or a security role nested within this default global role may result in problems with the WebLogic Security Service.

- For a security policy on a deployable resource (such as an application, EJB module, Web application module, Connector module, or startup/shutdown class), use the `Deployer` global role.

Permissions for Starting and Shutting Down Servers

WebLogic Server provides two ways to start and shut down WebLogic Server instances (servers): the `weblogic.Server` command and the Node Manager. Because the underlying components for the `weblogic.Server` command and the Node Manager are different, the two commands use different authorization methods.

The following sections provide more information about the permissions for starting and shutting down servers:

- "Permissions for Using the weblogic.Server Command" on page 3-10
- "Permissions for Using the Node Manager" on page 3-10
- "Shutting Down a WebLogic Server Instance" on page 3-11

Permissions for Using the weblogic.Server Command

The `weblogic.Server` command, which starts a WebLogic Server instance, calls methods that are protected by a security policy on the Server resource. To use this command, you must satisfy the requirements of the security policy on the Server resource.

Some `weblogic.Server` arguments set attributes for MBeans. However, because these arguments modify an MBean before the server is in the `RUNNING` state, the security policy on the Server resource, not the protection on the MBean, is the authorizer. For example, a user in the `Operator` global role can use the `-Dweblogic.ListenPort` argument to change a server's default listen port, but once the WebLogic Server instance is running, this user cannot change the listen port value.

For more information about `weblogic.Server`, see "Using the weblogic.Server Command" on page 2-17.

Permissions for Using the Node Manager

The Node Manager uses both MBeans and the security policy on the Server resource to start a remote server.

If you have configured a Node Manager on the host machine of a remote WebLogic Server instance, by default a user in the `Admin` or `Operator` global role can use the Node Manager to start the remote server.

For information about the Node Manager, refer to Managing Server Availability with Node Manager in the *Creating and Configuring WebLogic Server Domains* Guide.

Shutting Down a WebLogic Server Instance

Shutting down a WebLogic Server instance also involves both MBeans and the security policy on the Server resource. When a user issues a shutdown command, the server first determines whether that user is granted the `Admin` or `Operator` global role (per the MBean protection). Then, after the MBean operations run, the server determines whether the security policy on the Server resource authorizes the user to shut down the server.

CHAPTER

4 Using Log Messages to Manage WebLogic Server

WebLogic Server uses log messages to record information about events such as the deployment of new applications or the failure of one or more subsystems. The messages include information about the time and date of the event as well as the ID of the user who initiated the event.

You can view and sort these messages to detect problems, track down the source of a fault, and track system performance. For example, you can determine which user deployed a specific application or which user changed the thread pool count on a specific day. You can also create client applications that listen for these messages and respond automatically. For example, you can create an application that listens for messages indicating a failed subsystem. If a subsystem fails, the application can send email to a system administrator.

This topic contains the following sections:

For information on setting up your application to listen for messages, refer to the Using WebLogic Logging Services Guide.

WebLogic Server Log Messages

Compiled within the `weblogic.jar` file are sets of messages that each subsystem within WebLogic Server uses to communicate its status. For example, when you start a WebLogic Server instance, the Security subsystem writes a message to report its initialization status.

This section contains the following subsections:

- Message Attributes
- Message Output

Message Attributes

The messages for all subsystems contain a consistent set of fields (attributes) as described in the following table.

Table 4-1 Message Attributes

Attribute	Description
Timestamp	The time and date when the message originated, in a format that is specific to the locale.
Severity	Indicates the degree of impact or seriousness of the event reported by the message. For more information, refer to "Message Severity" on page 4-3.
Subsystem	Indicates the subsystem of WebLogic Server that was the source of the message. For example, EJB, RMI, JMS.

Table 4-1 Message Attributes

Attribute	Description
Server Name Machine Name Thread ID Transaction ID	Identify the origins of the message. Transaction ID is present only for messages logged within the context of a transaction.
User	The user ID under which the associated event was executed. To execute some pieces of internal code, WebLogic Server authenticates the ID of the user who initiates the execution and then runs the code under a special Kernel Identity user ID. J2EE modules such as EJBs that are deployed onto a server instance report the user ID that the module passes to the server. Log messages that are generated within a client JVM client do not include this field.
Message ID	A unique six-digit identifier. Message IDs through 499999 are reserved for WebLogic Server system messages.
Message Text	A description of the event or condition.

Message Severity

The **severity** attribute of a WebLogic Server log message indicates the potential impact of the event or condition that the message reports.

The following table lists the severity levels of log messages from WebLogic Server subsystems, starting from the lowest level of impact to the highest.

Table 4-2 Message Severity

Severity	Meaning
`INFO`	Used for reporting normal operations.
`WARNING`	A suspicious operation or configuration has occurred but it may not have an impact on normal operation.
`ERROR`	A user error has occurred. The system or application is able to handle the error with no interruption, and limited degradation, of service.

Table 4-2 Message Severity

Severity	Meaning
`NOTICE`	An `INFO` or `WARNING`-level message that is particularly important for monitoring the server.
`CRITICAL`	A system or service error has occurred. The system is able to recover but there might be a momentary loss, or permanent degradation, of service.
`ALERT`	A particular service is in an unusable state while other parts of the system continue to function. Automatic recovery is not possible; the immediate attention of the administrator is needed to resolve the problem.
`EMERGENCY`	The server is in an unusable state. This severity indicates a severe system failure or panic.

WebLogic Server subsystems generate many messages of lower severity and fewer messages of higher severity. For example, under normal circumstances, they generate many `INFO` messages and no `EMERGENCY` messages.

If your application uses the WebLogic logging services, it can use an additional severity level, `DEBUG`. WebLogic Server subsystems do not use this severity level. For more information, refer to Writing Debug Messages in the *Using WebLogic Logging Services* Guide.

Message Output

When a WebLogic Server instance outputs a message, the first line of each message begins with `####` followed by the message attributes. Each attribute is contained between angle brackets.

The following is an example of a log message:

```
####<Jun 2, 2002 10:23:02 AM PDT> <Info> <SSL> <bigbox> <myServer>
    <SSLListenThread> <harry> <> <004500> <Using exportable strength SSL>
```

In this example, the message attributes are: Timestamp, Severity, Subsystem, Machine Name, Server Name, Thread ID, User ID, Transaction ID, Message ID, and Message Text.

If a message is not logged within the context of a transaction, the angle brackets (separators) for Transaction ID are present even though no Transaction ID is present.

If the message includes a stack trace, the stack trace follows the list of message attributes.

The character encoding used in writing the log files is the default character encoding of the host system.

Exceptions and Stack Traces

Some WebLogic Server log messages indicate an exception has been thrown. If the JVM generates a stack trace with the exception, the WebLogic Server log message includes the stack trace.

You can specify whether a WebLogic Server instance writes the stack traces to its log file.

If an application that uses WebLogic logging services is running in a remote JVM, the application sends its exceptions and stack traces to the WebLogic logging services. You use the Administration Console to determine whether an instance of WebLogic Server writes these remote exceptions and stack traces to its log file.

For more information on configuring logging of exceptions and stack traces, refer to Configuring Debug Information in the Server Log File in the Administration Console Online Help.

WebLogic Server Log Files

To persist the messages that it generates, WebLogic Server writes the messages to log files. You can view these files with a standard text editor or the with log file viewer in the Administration Console.

Note: We recommend that you do not modify log files by manually editing them. Modifying a file changes the timestamp and can confuse log file rotation. In addition, editing a file might lock it and prevent updates from a WebLogic Server.

This section contains the following subsections:

- Local Log Files and Domain Log Files
- Log File Names and Locations
- Log File Rotation
- WebLogic Log File Viewer

Local Log Files and Domain Log Files

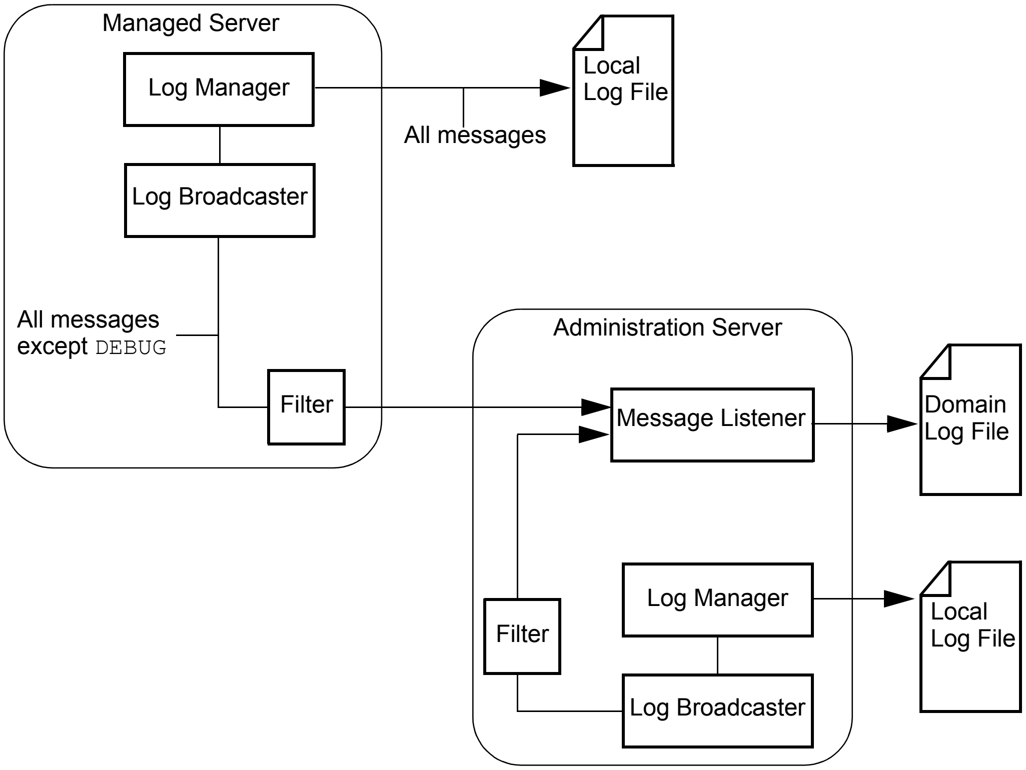
Each WebLogic Server instance writes all messages from its subsystems and applications to a log file that is located on the local host machine. In addition, each instance uses Java Management Extensions (JMX) to broadcast its messages as JMX notifications. A server broadcasts all messages and message text except for the following:

- Messages of the `DEBUG` severity level.
- Any stack traces that are included in a message.

When a WebLogic Server instance starts, the Administration Server's message listener registers itself with the server’s log broadcaster. At the time of registration, a default filter is provided that determines which messages the Administration Server listens for. The Administration Server writes these messages to an additional domain-wide log file. (See Figure 4-1.)

Note: If a Managed Server is running in Managed Server Independence (MSI) mode, it writes to the domain log file directly. See "MSI Mode and the Domain Log File."

Figure 4-1 WebLogic Server Logging Services



The default filter selects only messages of severity level `ERROR` and higher. In this way, the domain log provides a summary of the domain's overall status.

For any given WebLogic Server instance, you can override the default filter and create a domain log filter that causes a different set of messages to be written to the domain log file. For information on setting up a domain log filter for a WebLogic Server instance, refer to Domain Log Filters in the Administration Console Online Help.

If the Administration Server is unavailable, Managed Servers continue to write messages to their local log files, but they do not keep track of which messages they generate while the Administration Server is unavailable. For example, if the Administration Server is unavailable for two hours and then is restored, the domain log will not contain any messages that were generated during the two hours.

Log File Names and Locations

By default, the local server log file is named `./`*`SERVER_NAME`*`/`*`SERVER_NAME`*`.log`, where *`SERVER_NAME`* is the name of the server. The path is relative to the server's root directory.

The default name for a domain log file is `./`*`DOMAIN_NAME`*`.log`, where *`DOMAIN_NAME`* is the name of the domain. The path is relative to the root directory of the Administration Server.

To include a time or date stamp in the file name when the log file is rotated add `java.text.SimpleDateFormat` variables to the log's file name. For more information, see the next section, "Log File Rotation."

For information, see the following: on changing the names and locations of the log files, refer to the following topics in the Administration Console Online Help:

- "A Server's Root Directory" on page 2-30
- "Specifying General Log File Settings" in the Administration Console Help
- "Specifying the Name and Location of the Domain Log File" in the Administration Console Help

Log File Rotation

By default, local log files and domain log files grow in size indefinitely. You can specify that a WebLogic Server instance renames (rotates) a log file periodically. Old messages remain in the renamed log file and new messages accumulate in the new log file. You can base log file rotation on the size of the log file or on a time interval.

To include a time or date stamp in the file name when the log file is rotated, add `java.text.SimpleDateFormat` variables to the log's file name. Surround each variable with percentage (`%`) characters.

For example, if you enter the following value for a server's local log file name:
`myserver_%yyyy%_%MM%_%dd%_%hh%_%mm%.log`

the server's log file will be named:
`myserver_yyyy_MM_dd_hh_mm.log`

When the server instance rotates the log file, the rotated file name contains the date stamp. For example, if the server instance rotates its local log file on 2 April, 2003 at 10:05 AM, the log file that contains the old log messages will be named: `myserver_2003_04_02_10_05.log`

If you do not include a time and date stamp, the rotated log files are numbered in order of creation `*filename*nnnnn`, where `*filename*` is the name configured for the log file. For example: `myserver.log00007`.

You use the Administration Console to specify rotation criteria for each WebLogic Server instance's local log file. You also use the Administration Console to specify criteria for rotating the domain message log file.

You can also specify the maximum number of rotated files that can accumulate. After the number of log files reaches this number, subsequent file rotations overwrite the oldest log file.

Note: Log rotation by time is based on the timestamp of the files. Modifying a log file changes the timestamp and can confuse log rotation.

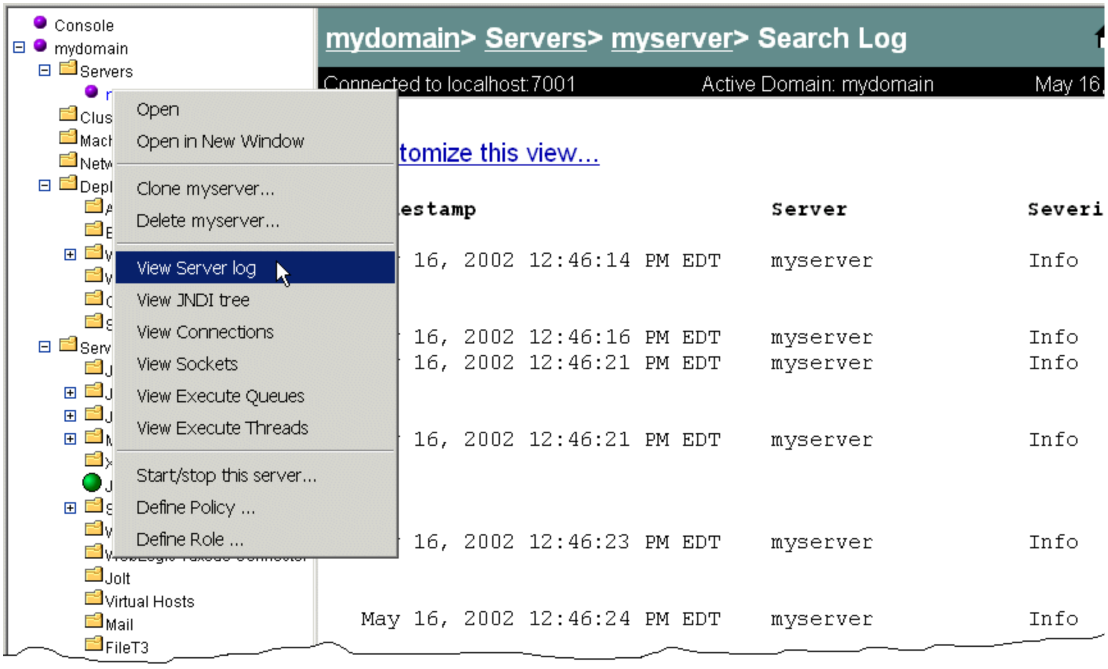
For information on specifying rotation criteria, refer to the following sections in the Administration Console Online Help:

- Specifying Log File Rotation
- Specifying Criteria for Rotating Domain Log Files

WebLogic Log File Viewer

The Administration Console provides separate but similar log viewers for the local server log and the domain-wide message log. The log viewer can search for messages based on fields within the message. For example, it can find and display messages based on the severity, time of occurrence, user ID, subsystem, or the short description. It can also display messages as they are logged, or search for past log messages.

Figure 4-2 Log Viewer in the Administration Console



```
estamp                          Server      Severi
 16, 2002 12:46:14 PM EDT      myserver    Info

 16, 2002 12:46:16 PM EDT      myserver    Info
 16, 2002 12:46:21 PM EDT      myserver    Info

 16, 2002 12:46:21 PM EDT      myserver    Info

 16, 2002 12:46:23 PM EDT      myserver    Info

May 16, 2002 12:46:24 PM EDT   myserver    Info
```

For information about viewing, configuring, and searching message logs from the Administration Console, refer to the following topics in the Administration Console Online Help:

- Viewing Server Logs
- Viewing the Domain Log

Because log files are simple text files, you can also use other applications to view them. For information about finding the log files, refer to "Log File Names and Locations" on page 4-8.

Output to Standard Out

In addition to writing messages to log files, a WebLogic Server instance can print a subset of its messages to standard out. By default, all messages of `ERROR` severity or higher are printed to standard out and messages of the `DEBUG` severity are not printed to standard out. If you configure a server to print stack traces to its log file, the stack traces are also printed to standard out.

If you use the Node Manager to start a Managed Server, the Node Manager writes standard out and standard error messages to its log file. You can view these messages from the Administration Console on the Machine→Monitoring tab.

For more information, refer to the following topics:

- For information on determining which WebLogic Server messages are sent to standard out, refer to Specifying General Log File Settings in the Administration Console Online Help.
- For information on printing stack traces to a log file, refer to Configuring Debug Information in the Server Log File in the Administration Console Online Help.
- For information on viewing messages for a Managed Server that you start with the Node Manager, refer to Managed Server Log Files in the *Creating and Configuring WebLogic Server Domains* guide.
- For information on viewing the standard output stream for server instances that run as Windows services, refer to “Redirecting Standard Out and Standard Error to a File” on page 2-56.

Redirecting System.out and System.err to a File

In addition to the configurable set of log messages that a server instance prints to standard out, servlets can invoke `system.out.println` and the JVM within which a WebLogic Server instance runs can send messages to standard error and standard out. If you use a WebLogic Server script to start a server instance, there is no default, persistent storage for the standard error and standard out messages.

The JVM within which a WebLogic Server instance runs also can send messages to standard error and standard out. If you use a WebLogic Server script to start a server instance, there is no default, persistent storage for the standard error and standard out messages.

If you want to keep a record of these messages, edit the WebLogic Server start script so it specifies the following Java options:

```
-Dweblogic.Stdout="stdout-filename"
-Dweblogic.Stderr="stderr-filename"
```

Where `stdout-filename` is the name of a file that you want to save standard out messages and `stderr-filename` is the name of a file that you want to save standard error messages.

To view the contents of these files, use a text editor or command prompt utility such as the DOS `tail` program. You cannot view them from the Administration Console.

Note: WebLogic Server prompts for entering your username and password are sent to standard out. If you use `-Dweblogic.Stdout`, you will no longer see the prompts to enter your username and password. To bypass this prompt, use a boot identity file as described in "Bypassing the Prompt for Username and Password" on page 2-7.

For information on passing arguments to the `weblogic.Server` command, refer to "Frequently Used Optional Arguments" on page 2-20.

Garbage Collection Comments

While the `-Dweblogic.Stdout` and `-Dweblogic.Stderr` options cause a JVM to redirect all of its `java.lang.System.out` and `java.lang.System.err` messages to a file, a JVM does not print its garbage collection comments to `System.out` or `System.err`. If you start a JVM with the `-verbosegc` option, the JVM will print the `verbosegc` output to the shell in which the JVM is running, regardless of whether you specify `-Dweblogic.Stdout` or `-Dweblogic.Stderr`. Some JVMs provide non-standard options for printing garbage collection comments to a file. For more information, view the help for your JVM's non-standard options by entering `java -X` in a shell.

Configuration Auditing

You can configure the Administration Server to emit log messages when a user changes the configuration or invokes management operations on any resource within a domain. For example, if a user disables SSL on a Managed Server in a domain, the Administration Server emits log messages. These messages provide an audit trail of changes within a domain's configuration (configuration auditing).

The following sections describe configuration auditing:

- "Enabling Configuration Auditing" on page 4-13
- "Configuration Auditing Messages" on page 4-14

The Administration Server writes configuration auditing messages to its local log file. Because all configuration auditing messages are of the `Info` severity, they are not written to the domain-wide message log by default. For information on changing this default, see "Domain Log Filters" in the Administration Console Online Help.

In addition to writing messages to its local log file, the Administration Server broadcasts configuration auditing messages as JMX notifications. You can create a JMX listener and filter that responds to these messages. For example, if the Administration Server emits a message that indicates an unauthorized user has attempted to change the domain's configuration, the JMX listener and filter can send email. See "Listening for Configuration Auditing Messages" in *Programming WebLogic Management Services with JMX.*

Enabling Configuration Auditing

To enable the Administration Server to emit configuration-auditing messages, do one of the following:

- Start the Administration Console and enable configuration auditing.

  See "Enabling Configuration Auditing" in the Administration Console Online Help.

- When you start the Administration Server, include the following Java option in the `weblogic.Server` command:

```
-Dweblogic.AdministrationMBeanAuditingEnabled=true
```

See "Using the weblogic.Server Command" on page 2-17.

- After the Administration Server has started, use the `weblogic.Admin` utility to change the value of the `DomainMBean`'s `AdministrationMBeanAuditingEnabled` attribute.

  For example, the following command disables configuration auditing for the `examples` domain:

  ```
  java weblogic.Admin SET
  -mbean examples:Name=examples,Type=Domain
  -property AdministrationMBeanAuditingEnabled true
  ```

  For information about using `weblogic.Admin` to change values of MBean attributes, "MBean Management Command Reference."

Configuration Auditing Messages

All configuration auditing messages are of the `Info` severity and are identified by message IDs that fall within the range of `159900-159910`.

The messages use managed bean (MBean) object names to identify resources. MBean object names provide an unambiguous identification regardless of the interface (Administration Console, command-line utility, or API) that is used to invoke operations or modify the resource. See "Using WebLogicObjectNames for WebLogic Server MBeans" in *Programming WebLogic Management Services with JMX.*

Table 4-3 summarizes the messages.

Table 4-3 Summary of Configuration Auditing Messages

When This Event Occurs...	WebLogic Server Generates a Message With This ID...	And This Message Text...
Authorized user creates a resource.	`159900`	`USER` *`username`* `CREATED` *`MBean-name`* where *`username`* identifies the WebLogic Server user who logged in and created a resource.

Table 4-3 Summary of Configuration Auditing Messages

When This Event Occurs...	WebLogic Server Generates a Message With This ID...	And This Message Text...
Unauthorized user attempts to create a resource.	`159901`	`USER` *`username`* `CREATED` *`MBean-name`* `FAILED weblogic.management. NoAccessRuntimeException:` *`exception-text stack-trace`* where *`username`* identifies the unauthorized WebLogic Server user.
Authorized user deletes a resource.	`159902`	`USER` *`username`* `REMOVED` *`MBean-name`* where *`username`* identifies the WebLogic Server user who logged in and created a resource.
Unauthorized user attempts to delete a resource.	`159903`	`USER` *`username`* `REMOVE` *`MBean-name`* `FAILED weblogic.management. NoAccessRuntimeException:` *`exception-text stack-trace`* where *`username`* identifies the WebLogic Server user who logged in and created a resource.

Authorized user changes a resource's configuration.	`159904`	`USER` *`username`* `MODIFIED` *`MBean-name`* `ATTRIBUTE` *`attribute-name`*
		`FROM` *`old-value`* `TO` *`new-value`*
		where *`username`* identifies the WebLogic Server user who logged in and changed the resource's configuration.

Unauthorized user attempts to change a resource's configuration.	`159905`	`USER` *`username`* `MODIFY` *`MBean-name`* `ATTRIBUTE` *`attribute-name`*
		`FROM` *`old-value`* `TO` *`new-value`*
		`FAILED weblogic.management. NoAccessRuntimeException:` *`exception-text stack-trace`*
		where *`username`* identifies the unauthorized WebLogic Server user.

Table 4-3 Summary of Configuration Auditing Messages

When This Event Occurs...	WebLogic Server Generates a Message With This ID...	And This Message Text...

Authorized user invokes an operation on a resource.
For example, a user deploys an application or starts a server instance.
	`159907`	`USER` *`username`* `INVOKED ON` *`MBean-name`*
		`METHOD` *`operation-name`*
		`PARAMS` *`specified-parameters`*
		where *`username`* identifies the WebLogic Server user who logged in and invoked a resource operation.

Unauthorized user attempts to invoke an operation on a resource.	`159908`	`USER` *`username`* `INVOKED ON` *`MBean-name`*
		`METHOD` *`operation-name`*
		`PARAMS` *`specified-parameters`*
		`FAILED weblogic.management. NoAccessRuntimeException:` *`exception-text stack-trace`*
		where *`username`* identifies the unauthorized WebLogic Server user.

Authorized user enables configuration auditing.	`159909`	`USER` *`username`*`, Configuration Auditing is enabled` where *`username`* identifies the WebLogic Server user who enabled configuration auditing.
Authorized user disables configuration auditing.	`159910`	`USER` *`username`*`, Configuration Auditing is disabled` where *`username`* identifies the WebLogic Server user who disabled configuration auditing.

Note: Each time an authorized user adds, modifies, or deletes a resource the Management subsystem also generates `Info` message with the ID `140009`. For example:

```
<Sep 15, 2003 11:54:47 AM EDT> <Info> <Management> <140009>
<Configuration changes for domain saved to the repository.>
```

The Management subsystem generates this message regardless of whether configuration auditing is enabled.

While the message informs you that the domain's configuration has changed, it does not provide the detailed information that configuration auditing messages provide. Nor does the Management subsystem generate this message when you invoke operations on resources.

Table 4-4 lists additional message attributes for configuration auditing messages. All configuration auditing messages specify the same values for these attributes.

Table 4-4 Common Message Attributes and Values

Message Attribute	Attribute Value
Severity	`Info`
Subsystem	`Configuration Audit`
User ID	`kernel identity` This value is always `kernel identity`, regardless of which user modified the resource or invoked the resource operation.
Server Name	*`AdminServerName`* Because the Administration Server maintains the configuration data for all resources in a domain, this value is always the name of the Administration Server.
Machine Name	*`AdminServerHostName`* Because the Administration Server maintains the configuration data for all resources in a domain, this value is always the name of the Administration Server's host machine.
Thread ID	*`execute-thread`* The value depends on the number of execute threads that are currently running on the Administration Server.
Timestamp	*`timeStamp`* at which the message is generated.

Additional Log Files

The log messages and files that are discussed in previous sections of this topic communicate events and conditions that affect the operation of the server or the application.

Some subsystems maintain additional log files to provide an audit of the subsystem's interactions under normal operating conditions. The following list describes each of the additional log files:

- The HTTP subsystem can keep a log of all HTTP transactions in a text file. You can set the attributes that define the behavior of HTTP access logs for each server or for each virtual host that you define. For more information, refer to "Setting Up HTTP Access Logs" on page 6-15.
- The JDBC subsystem keeps a log of various events related to JDBC connections, including registering JDBC drivers and SQL exceptions. Note that some events related to JDBC are written to the server log, such as when connections are created or refreshed or when configuration changes are made to JDBC objects.
- The JTA subsystem keeps a transaction log to report statistics on transactions. For more information, refer to "Monitoring and Logging Transactions" on page 7-10.

5 Deploying Applications

The following sections discuss installation and deployment of applications and application components on WebLogic Server:

- Supported Formats for Deployment
- Deploying a Web Application Using the (deprecated) weblogic.deploy Utility

Supported Formats for Deployment

J2EE applications and components can be deployed on WebLogic Servers as Enterprise Application Archive (EAR) files or in exploded directory format. However, if a J2EE application is deployed in exploded format, we recommend that no component other than the Web application component should be in exploded format. If the application is deployed in archived format, then we recommend that all of the components of the application also be in archived format.

Archived components may be EJB archives (JARs), Web Application Archives (WARs), or Resource Adaptor Archives (RAR).

For information about deploying J2EE Applications and an overview of WebLogic Server deployment, see WebLogic Server Application Deployment at *{DOCROOT}/programming/deploying.html*.

For information about deploying Web Applications see Assembling and Configuring Web Applications at *http://e-docs.bea.com/wls/docs70/webapp/index.html*.

For information about deploying Resource Adaptors, see Packaging and Deploying Resource Adapters at *http://e-docs.bea.com/wls/docs70/jconnector/packdepl.html*.

For information about deploying EJBs, see Packaging EJBs for the WebLogic Server Container at *http://e-docs.bea.com/wls/docs70*/ejb/EJB_packaging.html#1011066.

Deploying a Web Application Using the (deprecated) weblogic.deploy Utility

The `weblogic.Deployer` utility is new in WebLogic Server 7.0, and replaces the earlier `weblogic.deploy` utility, which has been deprecated. Please note that all utilities and APIs using the older, WebLogic Server 6.x deployment protocol are now deprecated. Use the WebLogic Server 7.0 two-phase deployment tools and utilities for all application deployment.

For information about `weblogic.Deployer`, see weblogic.Deployer Utility at *http://e-docs.bea.com/wls/docs70*/ejb/EJB_packaging.html#1011066.

To deploy a Web Application using the `weblogic.deploy` utility:

1. Set up your local environment so that WebLogic Server classes are in your system `CLASSPATH` and the JDK is available. You can use the `setEnv` script located in the `config/`*`mydomain`* directory to set the `CLASSPATH`.

2. Enter the following command:

```
% java weblogic.deploy -port port_number -host host_name
   -user username -component application:target deploy
   password name application source
```

   Where:

   - *`port_number`* is the port number where WebLogic Server is listening for requests
   - *`host_name`* is the name of the machine hosting WebLogic Server
   - *`username`* is a user that has permission to complete the request on the server that you specify in the `-host` argument. The default is the username that you

used to start the server that you specify in the `-host` argument.username is the user account under which WebLogic Server is booted.

For information about permissions for system administration tasks, refer to "Protecting System Administration Operations" on page 3-1.

- *`application`* is the name you want to assign to this Web Application.
- *`target`* is the name of a server, cluster or virtual host to be targeted by this Web Application. You can enter multiple targets, separated by a comma.
- *`password`* is your system administration password
- *name* is your system administration name
- *`source`* is the full pathname of the `.war` file you want to deploy, or the full pathname of a directory containing a Web Application in exploded directory format.

For example:

```
java weblogic.deploy -port 7001 -host myhost -component
   myWebApp:myserver deploy pswd1234 myWebApp d:\myWebApp.war
```

Deployment Documentation

WebLogic Server Deployment at http://e-docs.bea.com/wls/docs70/programming/deploying.html describes Weblogic Server deployment and deployment tools, procedures, and best practices. :

Table 5-1 Deployment Documents

Document	Deployment Topics
WebLogic Builder	How to use WebLogic Builder to edit and generate XML deployment descriptor files for J2EE applications and their components.
Developing WebLogic Server Applications	How to deploy WebLogic Server J2EE applications.

Table 5-1 Deployment Documents

Document	Deployment Topics
Administration Console Online Help	How to use the Administration Console for deployment tasks.
Programming WebLogic EJBs	How to deploy WebLogic Server EJBs.
Programming WebLogic J2EE Connectors	How to deploy WebLogic Server J2EE Connectors.
Assembling and Configuring Web Applications	How to deploy Weblogic Server Web Applications.
Programming WebLogic JSP	How to deploy applets from JSP.
Understanding Cluster Configuration and Application Deployment	How to deploy to clustered servers.
WebLogic Server Application Packaging and Classloading	How to package WebLogic Server application components.

CHAPTER

6 Configuring WebLogic Server Web Components

The following sections discuss how to configure WebLogic Server Web components:

Overview

In addition to its ability to host dynamic Java-based distributed applications, WebLogic Server is also a fully functional Web server that can handle high volume Web sites, serving static files such as HTML files and image files as well as servlets and JavaServer Pages (JSP). WebLogic Server supports the HTTP 1.1 standard.

HTTP Parameters

You can configure the HTTP operating parameters using the Administration Console for each Server or Virtual Host.

Attribute	Description	Range of Values	Default Value
Default Server Name	When WebLogic Server redirects a request, it sets the host name returned in the HTTP response header with the string specified with Default Server Name. Useful when using firewalls or load balancers and you want the redirected request from the browser to reference the same host name that was sent in the original request.	String	Null

Enable Keepalives	This attribute sets whether or not HTTP keep-alive is enabled	Boolean
		True = enabled
		False = not enabled
			True

Attribute	Description	Range of Values	Default Value

Send Server Header	If set to false, the server name is not sent with the HTTP response. Useful for wireless applications where there is limited space for headers.	Boolean
		True = enabled
		False = not enabled
			True

Duration (labeled Keep Alive Secs on the Virtual Host panel)	The number of seconds that WebLogic Server waits before closing an inactive HTTP connection.	Integer	30
HTTPS Duration (labeled Https Keep Alive Secs on the Virtual Host panel)	The number of seconds that WebLogic Server waits before closing an inactive HTTPS connection.	Integer	60
Post Timeout Secs	This attribute sets the timeout (in seconds) that WebLogic Server waits between receiving chunks of data in an HTTP POST data. Used to prevent denial-of-service attacks that attempt to overload the server with POST data. Max Post Time -1 Max Post Size -1	Integer	30
Max Post Time	This attribute sets the time (in seconds) that WebLogic Server waits for chunks of data in an HTTP POST data.	Integer	0
Max Post Size	This attribute sets the size of the maximum chunks of data in an HTTP POST data.	Integer	0

Attribute	Description	Range of Values	Default Value
External DNS Address	If your system includes an address translation firewall that sits between the clustered WebLogic Servers and a plug-in to a web server front-end, such as the Netscape (proxy) plug-in, set this attribute to the address used by the plug-in to talk to this server.		

Configuring the Listen Port

You can specify the port that each WebLogic Server listens on for HTTP requests. Although you can specify any valid port number, if you specify port `80`, you can omit the port number from the HTTP request used to access resources over HTTP. For example, if you define port `80` as the listen port, you can use the form `http://hostname/myfile.html` instead of `http://hostname:portnumber/myfile.html`.

Note: Setting up WebLogic Server to listen on port 80

- weblogic.system.listenPort=integer
- weblogic.system.enableSetUID=boolean
- weblogic.system.nonPrivUser=UNIXusername
- weblogic.system.enableSetGID=boolean
- weblogic.system.nonPrivGroup=UNIXgroupname

The latter four properties apply only to UNIX users.

On UNIX systems, binding a process to a port less than 1025 must be done from the account of a privileged user, usually root. Consequently, if you want WebLogic Server to listen on port 80, you must start WebLogic Server as a privileged user; yet it is generally considered undesirable from a security standpoint to allow long-running processes like WebLogic Server to run with more privileges than necessary. WebLogic needs root privileges only until the port is bound.

By setting the weblogic.system.enableSetUID property (and, if desired, the weblogic.system.enableSetGID property) to true, you enable an internal process by which WebLogic Server switches its UNIX user ID (UID) after it binds to port 80. The companion properties, weblogic.system.nonPrivUser and weblogic.system.nonPrivGroup, identifies a non-privileged UNIX user account (and optionally a groupname) under which WebLogic will run after startup.

You may choose to switch to the UNIX account "nobody," which is the least privileged user on most UNIX systems. If desired, you may create a UNIX user account expressly for running WebLogic Server. Make sure that files needed by WebLogic Server, such as log files and the WebLogic classes, are accessible by the non-privileged user. Once ownership of the WebLogic process has switched to the non-privileged user, WebLogic will have the same read, write, and execute permissions as the non-privileged user.

You define a separate listen port for regular and secure (using SSL) requests. You define the regular listen port on the Servers node in the Administration Console, under the Configuration/General tab, and you define the SSL listen port under the Connections/SSL tab.

Web Applications

HTTP and Web Applications are deployed according to the Servlet 2.3 specification from Sun Microsystems, which describes the use of *Web Applications* as a standardized way of grouping together the components of a Web-based application. These components include JSP pages, HTTP servlets, and static resources such as HTML pages or image files. In addition, a Web Application can access external resources such as EJBs and JSP tag libraries. Each server can host any number of Web Applications. You normally use the name of the Web Application as part of the URI you use to request resources from the Web Application.

By default JSPs are compiled into the servers' temporary directory the location for which is (for a server: "myserver" and for a webapp: "mywebapp"):
<domain_dir>\myserver\.wlnotdelete\appname_mywebapp_4344862

For more information, see Assembling and Configuring Web Applications at `http://e-docs.bea.com/wls/docs70/webapp/index.html`.

Web Applications and Clustering

Web Applications can be deployed in a cluster of WebLogic Servers. When a user requests a resource from a Web Application, the request is routed to one of the servers of the cluster that host the Web Application. If an application uses a session object, then sessions must be replicated across the nodes of the cluster. Several methods of replicating sessions are provided.

For more information, see Using WebLogic Server Clusters at http://e-docs.bea.com/wls/docs70/cluster/index.html.

Designating a Default Web Application

Every server and virtual host in your domain can declare a *default Web Application*. The default Web Application responds to any HTTP request that cannot be resolved to another deployed Web Application. In contrast to all other Web Applications, the default Web Application does *not* use the Web Application name as part of the URI. Any Web Application targeted to a server or virtual host can be declared as the default Web Application. (Targeting a Web Application is discussed later in this section. For more information about virtual hosts, see "Configuring Virtual Hosting" on page 6-7).

The examples domain that is shipped with Weblogic Server has a default Web Application already configured. The default Web Application in this domain is named `DefaultWebApp` and is located in the `applications` directory of the domain.

If you declare a default Web Application that fails to deploy correctly, an error is logged and users attempting to access the failed default Web Application receive an HTTP `400` error message.

For example, if your Web Application is called `shopping`, you would use the following URL to access a JSP called `cart.jsp` from the Web Application:

```
http://host:port/shopping/cart.jsp
```

If, however, you declared `shopping` as the default Web Application, you would access `cart.jsp` with the following URL:

```
http://host:port/cart.jsp
```

(Where `host` is the host name of the machine running WebLogic Server and `port` is the port number where the WebLogic Server is listening for requests.)

To designate a default Web Application for a server or virtual host, use the Administration Console:

1. In the left pane, expand the Web Application node
2. Select your Web Application
3. In the right pane, select the Targets tab.
4. Select the Servers tab and move the server (or virtual host) to the chosen column. (You can also target all the servers in a cluster by selecting the Clusters tab and moving the cluster to the Chosen column.)
5. Click Apply.
6. Expand the Servers (or virtual host) node in the left pane.
7. Select the appropriate server or virtual host.
8. In the right pane, select the Connections tab
9. Select the HTTP tab (If you are configuring a virtual host, select the General tab instead.)
10. Select a Web Application from the drop-down list labeled Default Web Application.
11. Click Apply.
12. If you are declaring a default Web Application for one or more managed servers, repeat these procedures for each managed server.

Configuring Virtual Hosting

Virtual hosting allows you to define host names that servers or clusters respond to. When you use virtual hosting you use DNS to specify one or more host names that map to the IP address of a WebLogic Server or cluster and you specify which Web

Applications are served by the virtual host. When used in a cluster, load balancing allows the most efficient use of your hardware, even if one of the DNS host names processes more requests than the others.

For example, you can specify that a Web Application called `books` responds to requests for the virtual host name `www.books.com`, and that these requests are targeted to WebLogic Servers A,B and C, while a Web Application called `cars` responds to the virtual host name `www.autos.com` and these requests are targeted to WebLogic Servers D and E. You can configure a variety of combinations of virtual host, WebLogic Servers, clusters and Web Applications, depending on your application and Web server requirements.

Virtual hosting also allows you to create one directory to serve static files such as images for multiple Web Applications. For example, to serve image files using a virtual host, you can create a mapping similar to the folowing:

```
<virtual-directory-mapping>

  <local-path>c:/usr/gifs</local-path>

  <url-pattern>/images/*</url-pattern>

</virtual-directory-mapping>
```

A request to `HTTP:// localhost:7001/mywebapp/images/test.gif` will cause your WebLogic Server implementation to look for the requested image at: `c:/usr/gifs/images/*`.

This use of a virtual host requires you to create a directory named images to hold the image files. This directory must be located in the relative uri, such as "`/images/test.gif`".

For each virtual host that you define you can also separately define HTTP parameters and HTTP access logs. The HTTP parameters and access logs set for a *virtual host* override those set for a *server*. You may specify any number of virtual hosts.

You activate virtual hosting by targeting the virtual host to a server or cluster of servers. Virtual hosting targeted to a cluster will be applied to all servers in the cluster.

Virtual Hosting and the Default Web Application

You can also designate a *default Web Application* for each virtual host. The default Web Application for a virtual host responds to all requests that cannot be resolved to other Web Applications deployed on same server or cluster as the virtual host.

Unlike other Web Applications, a default Web Application does not use the Web Application name (also called the *context path*) as part of the URI used to access resources in the default Web Application.

For example, if you defined virtual host name `www.mystore.com` and targeted it to a server on which you deployed a Web Application called `shopping`, you would access a JSP called `cart.jsp` from the `shopping` Web Application with the following URI:

```
http://www.mystore.com/shopping/cart.jsp
```

If, however, you declared `shopping` as the default Web Application for the virtual host `www.mystore.com`, you would access `cart.jsp` with the following URI:

```
http://www.mystore.com/cart.jsp
```

For more information, see "How WebLogic Server Resolves HTTP Requests" on page 6-11.

Setting Up a Virtual Host

To define a virtual host, use the Administration Console to:

1. Create a new Virtual Host.
   a. Expand the Services node in the left pane. The node expands and displays a list of services.
   b. Click on the virtual hosts node. If any virtual hosts are defined, the node expands and displays a list of virtual hosts.
   c. Click on Create a New Virtual Host in the right pane.
   d. Enter a name to represent this virtual host.

   e. Enter the virtual host names, one per line. Only requests matching one of these virtual host names will be handled by the WebLogic Server or cluster targeted by this virtual host.

   f. (Optional) Assign a default Web Application to this virtual host.

   g. Click Create.

2. Define logging and HTTP parameters:

   a. (Optional) Click on the Logging tab and fill in HTTP access log attributes (For more information, see "Setting Up HTTP Access Logs" on page 6-15.)

   b. Select the HTTP tab and fill in the HTTP Parameters.

3. Define the servers that will respond to this virtual host.

   a. Select the Targets tab.

   b. Select the Servers tab. You will see a list of available servers.

   c. Select a server in the available column and use the right arrow button to move the server to the chosen column.

4. Define the clusters that will respond to this virtual host (optional). You must have previously defined a WebLogic Cluster. For more information, see Using WebLogic Server Clusters at `http://e-docs.bea.com/wls/docs70/cluster/index.html`.

   a. Select the Targets tab.

   b. Select the Clusters tab. You will see a list of available servers.

   c. Select a cluster in the available column and use the right arrow button to move the cluster to the chosen column. The virtual host is targeted on all of the servers of the cluster.

5. Target Web Applications to the virtual host.

   a. Click the Web Applications node in the left panel.

   b. Select the Web Application you want to target.

   c. Select the Targets tab in the right panel.

   d. Select the Virtual Hosts tab.

e. Select Virtual Host in the available column and use the right arrow button to move the Virtual Host to the chosen column.

You must add a line naming the virtual host to the etc/hosts file on your server to ensure that the virtual host name can be resolved.

How WebLogic Server Resolves HTTP Requests

When WebLogic Server receives an HTTP request, it resolves the request by parsing the various parts of the URL and using that information to determine which Web Application and/or server should handle the request. The examples below demonstrate various combinations of requests for Web Applications, virtual hosts, servlets, JSPs, and static files and the resulting response.

Note: If you package your Web Application as part of an Enterprise Application, you can provide an alternate name for a Web Application that is used to resolve requests to the Web Application. For more information, see Deploying Web Applications as Part of an Enterprise Application at `http://e-docs.bea.com/wls/docs70/webapp/deployment.html#war-ear`.

The table below provides some sample URLs and the file that is served by WebLogic Server. The Index Directories Checked column refers to the Index Directories attribute that controls whether or not a directory listing is served if no file is specifically requested. You set Index Directories using the Administration Console, on the Web Applications node, under the Configuration →Files tab.

Table 6-1 Examples of How WebLogic Server Resolves URLs

URL	Index Directories Checked?	This file is served in response
`http://host:port/apples`	No	Welcome file* defined in the `apples` Web Application.
`http://host:port/apples`	Yes	Directory listing of the top level directory of the `apples` Web Application.
`http://host:port/oranges/naval`	Does not matter	Servlet mapped with `<url-pattern>` of `/naval` in the `oranges` Web Application. There are additional considerations for servlet mappings. For more information, see Configuring Servlets at `http://e-docs.bea.com/wls/docs70/webapp/components.html#configuring-servlets`.

Table 6-1 Examples of How WebLogic Server Resolves URLs

URL	Index Directories Checked?	This file is served in response
http://host:port/naval	Does not matter	Servlet mapped with `<url-pattern>` of `/naval` in the `oranges` Web Application and `oranges` is defined as the default Web Application. For more information, see Configuring Servlets at `http://e-docs.bea.com/wls/docs70/webapp/components.html#configuring-servlets`.
http://host:port/apples/pie.jsp	Does not matter	`pie.jsp`, from the top-level directory of the `apples` Web Application.
`http://host:port`	Yes	Directory listing of the top level directory of the *default* Web Application
`http://host:port`	No	Welcome file* from the *default* Web Application.
`http://host:port/apples/myfile.html`	Does not matter	`myfile.html`, from the top level directory of the `apples` Web Application.
`http://host:port/myfile.html`	Does not matter	`myfile.html`, from the top level directory of the *default* Web Application.
`http://host:port/apples/images/red.gif`	Does not matter	`red.gif`, from the images subdirectory of the top-level directory of the `apples` Web Application.

Table 6-1 Examples of How WebLogic Server Resolves URLs

URL	Index Directories Checked?	This file is served in response
`http://host:port/myFile.html` Where `myfile.html` does not exist in the `apples` Web Application and a *default servlet* has not been defined.	Does not matter	Error 404 For more information, see Customizing HTTP Error Responses at `http://e-docs.bea.com/wls/docs70/webapp/components.html#error-page`.
`http://www.fruit.com/`	No	Welcome file* from the default Web Application for a virtual host with a host name of `www.fruit.com`.
`http://www.fruit.com/`	Yes	Directory listing of the top level directory of the `default` Web Application for a virtual host with a host name of `www.fruit.com`.
`http://www.fruit.com/oranges/myfile.html`	Does not matter	`myfile.html`, from the `oranges` Web Application that is targeted to a virtual host with host name `www.fruit.com`.

* For more information, see Configuring Welcome Pages at `http://e-docs.bea.com/wls/docs70/webapp/components.html#welcome_pages`.

Setting Up HTTP Access Logs

WebLogic Server can keep a log of all HTTP transactions in a text file, in either *common log format* or *extended log format*. Common log format is the default, and follows a standard convention. Extended log format allows you to customize the information that is recorded. You can set the attributes that define the behavior of HTTP access logs for each server or for each virtual host that you define.

For information on setting up HTTP logging for a server or a virtual host, refer to the following topics in the Administration Console online help:

- Specifying HTTP Log File Settings for a Server
- Specifying HTTP Log File Settings for a Virtual Host

Log Rotation

You can also choose to rotate the log file based on either the size of the file or after a specified amount of time has passed. When either one of these two criteria are met, the current access log file is closed and a new access log file is started. If you do not configure log rotation, the HTTP access log file grows indefinitely. You can configure the name of the access log file to include a time and date stamp that indicates when the file was rotated. If you do not configure a time stamp, each rotated file name inlcudes a numeric portion that is incremented upon each rotation. Separate HTTP Access logs are kept for each Web Server you have defined.

Common Log Format

The default format for logged HTTP information is the common log format at `http://www.w3.org/Daemon/User/Config/Logging.html#common-logfile-format`. This standard format follows the pattern:

```
host RFC931 auth_user [day/month/year:hour:minute:second
    UTC_offset] "request" status bytes
```

where:

`host`
: Either the DNS name or the IP number of the remote client

`RFC931`
: Any information returned by IDENTD for the remote client; WebLogic Server does not support user identification

`auth_user`
: If the remote client user sent a userid for authentication, the user name; otherwise "-"

`day/month/year:hour:minute:second UTC_offset`
: Day, calendar month, year and time of day (24-hour format) with the hours difference between local time and GMT, enclosed in square brackets

`"request"`
: First line of the HTTP request submitted by the remote client enclosed in double quotes

`status`
: HTTP status code returned by the server, if available; otherwise "-"

`bytes`
: Number of bytes listed as the content-length in the HTTP header, not including the HTTP header, if known; otherwise "-"

Setting Up HTTP Access Logs by Using Extended Log Format

WebLogic Server also supports extended log file format, version 1.0, as defined by the W3C. This is an emerging standard, and WebLogic Server follows the draft specification from W3C at `www.w3.org/TR/WD-logfile.html`. The current definitive reference may be found from the W3C Technical Reports and Publications page at `www.w3.org/pub/WWW/TR`.

The extended log format allows you to specify the type and order of information recorded about each HTTP communication. To enable the extended log format, set the Format attribute on the HTTP tab in the Administration Console to `Extended`. (See "Creating Custom Field Identifiers" on page 6-19).

You specify what information should be recorded in the log file with directives, included in the actual log file itself. A directive begins on a new line and starts with a # sign. If the log file does not exist, a new log file is created with default directives. However, if the log file already exists when the server starts, it must contain legal directives at the head of the file.

Creating the Fields Directive

The first line of your log file must contain a directive stating the version number of the log file format. You must also include a `Fields` directive near the beginning of the file:

```
#Version: 1.0
#Fields: xxxx xxxx xxxx ...
```

Where each `xxxx` describes the data fields to be recorded. Field types are specified as either simple identifiers, or may take a prefix-identifier format, as defined in the W3C specification. Here is an example:

```
#Fields: date time cs-method cs-uri
```

This identifier instructs the server to record the date and time of the transaction, the request method that the client used, and the URI of the request for each HTTP access. Each field is separated by white space, and each record is written to a new line, appended to the log file.

Note: The `#Fields` directive must be followed by a new line in the log file, so that the first log message is not appended to the same line.

Supported Field identifiers

The following identifiers are supported, and do not require a prefix.

`date`
: Date at which transaction completed, field has type <date>, as defined in the W3C specification.

`time`
: Time at which transaction completed, field has type <time>, as defined in the W3C specification.

`time-taken`
: Time taken for transaction to complete in seconds, field has type <fixed>, as defined in the W3C specification.

`bytes`
: Number of bytes transferred, field has type <integer>.

Note that the `cached` field defined in the W3C specification is not supported in WebLogic Server.

The following identifiers require prefixes, and cannot be used alone. The supported prefix combinations are explained individually.

IP address related fields:
: These fields give the IP address and port of either the requesting client, or the responding server. This field has type <address>, as defined in the W3C specification. The supported prefixes are:

  `c-ip`
  : The IP address of the client.

  `s-ip`
  : The IP address of the server.

DNS related fields
: These fields give the domain names of the client or the server. This field has type <name>, as defined in the W3C specification. The supported prefixes are:

  `c-dns`
  : The domain name of the requesting client.

  `s-dns`
  : The domain name of the requested server.

`sc-status`
: Status code of the response, for example (404) indicating a “File not found” status. This field has type <integer>, as defined in the W3C specification.

`sc-comment`
: The comment returned with status code, for instance “File not found”. This field has type <text>.

`cs-method`
: The request method, for example GET or POST. This field has type <name>, as defined in the W3C specification.

`cs-uri`
: The full requested URI. This field has type <uri>, as defined in the W3C specification.

`cs-uri-stem`
: Only the stem portion of URI (omitting query). This field has type <uri>, as defined in the W3C specification.

`cs-uri-query`
: Only the query portion of the URI. This field has type <uri>, as defined in the W3C specification.

Creating Custom Field Identifiers

You can also create user-defined fields for inclusion in an HTTP access log file that uses the extended log format. To create a custom field you identify the field in the ELF log file using the `Fields` directive and then you create a matching Java class that generates the desired output. You can create a separate Java class for each field, or the Java class can output multiple fields. A sample of the Java source for such a class is included in this document. See "Java Class for Creating a Custom ELF Field" on page 6-23.

To create a custom field:

1. Include the field name in the `Fields` directive, using the form:

   `x-`*`myCustomField`*.

   Where *`myCustomField`* is a fully-qualified class name.

   For more information on the `Fields` directive, see "Creating the Fields Directive" on page 6-17.

2. Create a Java class with the same fully-qualified class name as the custom field you defined with the `Fields` directive (for example `myCustomField`). This class defines the information you want logged in your custom field. The Java class must implement the following interface:

   `weblogic.servlet.logging.CustomELFLogger`

   In your Java class, you must implement the `logField()` method, which takes a `HttpAccountingInfo` object and `FormatStringBuffer` object as its arguments:

   - Use the `HttpAccountingInfo` object to access HTTP request and response data that you can output in your custom field. Getter methods are provided to access this information. For a complete listing of these get methods, see "Get Methods of the HttpAccountingInfo Object" on page 6-20.

- Use the `FormatStringBuffer` class to create the contents of your custom field. Methods are provided to create suitable output. For more information on these methods, see the Javadocs for FormatStringBuffer (see `http://e-docs.bea.com/wls/docs70/javadocs/weblogic/servlet/logging/FormatStringBuffer.html`).

3. Compile the Java class and add the class to the `CLASSPATH` statement used to start WebLogic Server. You will probably need to modify the `CLASSPATH` statements in the scripts that you use to start WebLogic Server.

   Note: Do not place this class inside of a Web Application or Enterprise Application in exploded or jar format.

4. Configure WebLogic Server to use the extended log format. For more information, see "Setting Up HTTP Access Logs by Using Extended Log Format" on page 6-16.

Note: When writing the Java class that defines your custom field, you should not execute any code that is likely to slow down the system (For instance, accessing a DBMS or executing significant I/O or networking calls.) Remember, an HTTP access log file entry is created for *every* HTTP request.

Note: If you want to output more than one field, delimit the fields with a tab character. For more information on delimiting fields and other ELF formatting issues, see Extended Log Format at `http://www.w3.org/TR/WD-logfile-960221.html`.

Get Methods of the HttpAccountingInfo Object

The following methods return various data regarding the HTTP request. These methods are similar to various methods of `javax.servlet.ServletRequest`, `javax.servlet.http.Http.ServletRequest`, and `javax.servlet.http.HttpServletResponse`.

For details on these methods see the corresponding methods in the Java interfaces listed in the following table, or refer to the specific information contained in the table.

Table 6-2 Getter Methods of HttpAccountingInfo

`HttpAccountingInfo Methods`	**Where to find information on the methods**
`Object getAttribute(String name);`	javax.servlet.ServletRequest
`Enumeration getAttributeNames();`	javax.servlet.ServletRequest

Table 6-2 Getter Methods of HttpAccountingInfo

`HttpAccountingInfo Methods`	**Where to find information on the methods**
`String getCharacterEncoding();`	javax.servlet.ServletRequest
`int getResponseContentLength();`	javax.servlet.ServletResponse.setContentLength() This method *gets* the content length of the response, as set with the `setContentLength()` method.
`String getContentType();`	javax.servlet.ServletRequest
`Locale getLocale();`	javax.servlet.ServletRequest
`Enumeration getLocales();`	javax.servlet.ServletRequest
`String getParameter(String name);`	javax.servlet.ServletRequest
`Enumeration getParameterNames();`	javax.servlet.ServletRequest
`String[] getParameterValues(String name);`	javax.servlet.ServletRequest
`String getProtocol();`	javax.servlet.ServletRequest
`String getRemoteAddr();`	javax.servlet.ServletRequest
`String getRemoteHost();`	javax.servlet.ServletRequest
`String getScheme();`	javax.servlet.ServletRequest
`String getServerName();`	javax.servlet.ServletRequest
`int getServerPort();`	javax.servlet.ServletRequest
`boolean isSecure();`	javax.servlet.ServletRequest
`String getAuthType();`	javax.servlet.http.Http.ServletRequest
`String getContextPath();`	javax.servlet.http.Http.ServletRequest
`Cookie[] getCookies();`	javax.servlet.http.Http.ServletRequest
`long getDateHeader(String name);`	javax.servlet.http.Http.ServletRequest
`String getHeader(String name);`	javax.servlet.http.Http.ServletRequest

Table 6-2 Getter Methods of HttpAccountingInfo

`HttpAccountingInfo Methods`	**Where to find information on the methods**
`Enumeration getHeaderNames();`	javax.servlet.http.Http.ServletRequest
`Enumeration getHeaders(String name);`	javax.servlet.http.Http.ServletRequest
`int getIntHeader(String name);`	javax.servlet.http.Http.ServletRequest
`String getMethod();`	javax.servlet.http.Http.ServletRequest
`String getPathInfo();`	javax.servlet.http.Http.ServletRequest
`String getPathTranslated();`	javax.servlet.http.Http.ServletRequest
`String getQueryString();`	javax.servlet.http.Http.ServletRequest
`String getRemoteUser();`	javax.servlet.http.Http.ServletRequest
`String getRequestURI();`	javax.servlet.http.Http.ServletRequest
`String getRequestedSessionId();`	javax.servlet.http.Http.ServletRequest
`String getServletPath();`	javax.servlet.http.Http.ServletRequest
`Principal getUserPrincipal();`	javax.servlet.http.Http.ServletRequest
`boolean isRequestedSessionIdFromCookie();`	javax.servlet.http.Http.ServletRequest
`boolean isRequestedSessionIdFromURL();`	javax.servlet.http.Http.ServletRequest
`boolean isRequestedSessionIdFromUrl();`	javax.servlet.http.Http.ServletRequest
`boolean isRequestedSessionIdValid();`	javax.servlet.http.Http.ServletRequest
`String getFirstLine();`	Returns the first line of the HTTP request, for example: `GET /index.html HTTP/1.0`
`long getInvokeTime();`	Returns the length of time it took for the service method of a servlet to write data back to the client.
`int getResponseStatusCode();`	javax.servlet.http.HttpServletResponse

Table 6-2 Getter Methods of HttpAccountingInfo

`HttpAccountingInfo Methods`	**Where to find information on the methods**
`String getResponseHeader(String name);`	javax.servlet.http.HttpServletResponse

Listing 6-1 Java Class for Creating a Custom ELF Field

```
import weblogic.servlet.logging.CustomELFLogger;
import weblogic.servlet.logging.FormatStringBuffer;
import weblogic.servlet.logging.HttpAccountingInfo;

/* This example outputs the User-Agent field into a
 custom field called MyCustomField
*/

public class MyCustomField implements CustomELFLogger{

public void logField(HttpAccountingInfo metrics,
  FormatStringBuffer buff) {
  buff.appendValueOrDash(metrics.getHeader("User-Agent"));
  }
}
```

Preventing POST Denial-of-Service Attacks

A Denial-of-Service attack is a malicious attempt to overload a server with phony requests. One common type of attack is to send huge amounts of data in an HTTP `POST` method. You can set three attributes in WebLogic Server that help prevent this type of attack. These attributes are set in the console, under *Servers* or *virtual hosts*. If you define these attributes for a virtual host, the values set for the virtual host override those set under *Servers*.

`PostTimeoutSecs`

You can limit the amount of time that WebLogic Server waits between receiving chunks of data in an HTTP POST.

`MaxPostTimeSecs`

Limits the total amount of time that WebLogic Server spends receiving post data. If this limit is triggered, a `PostTimeoutException` is thrown and the following message is sent to the server log:

```
Post time exceeded MaxPostTimeSecs.
```

`MaxPostSize`

Limits the number of bytes of data received in a POST from a single request. If this limit is triggered, a `MaxPostSizeExceeded` exception is thrown and the following message is sent to the server log:

```
POST size exceeded the parameter MaxPostSize.
```

An HTTP error code 413 (Request Entity Too Large) is sent back to the client.

If the client is in listening mode, it gets these messages. If the client is not in listening mode, the connection is broken.

Setting Up WebLogic Server for HTTP Tunneling

HTTP tunneling provides a way to simulate a stateful socket connection between WebLogic Server and a Java client when your only option is to use the HTTP protocol. It is generally used to *tunnel* through an HTTP port in a security firewall. HTTP is a stateless protocol, but WebLogic Server provides tunneling functionality to make the connection appear to be a regular T3Connection. However, you can expect some performance loss in comparison to a normal socket connection.

Configuring the HTTP Tunneling Connection

Under the HTTP protocol, a client may only make a request, and then accept a reply from a server. The server may not voluntarily communicate with the client, and the protocol is stateless, meaning that a continuous two-way connection is not possible.

WebLogic HTTP tunneling simulates a T3Connection via the HTTP protocol, overcoming these limitations. There are two attributes that you can configure in the Administration Console to tune a tunneled connection for performance. You access these attributes in the Servers section, under the Connections and Protocols tabs. It is advised that you leave them at their default settings unless you experience connection problems. These properties are used by the server to determine whether the client connection is still valid, or whether the client is still alive.

`Enable Tunneling`

Enables or disables HTTP tunneling. HTTP tunneling is disabled by default.

Note that the server must also support both the HTTP and T3 protocols in order to use HTTP tunneling.

`Tunneling Client Ping`

When an HTTP tunnel connection is set up, the client automatically sends a request to the server, so that the server may volunteer a response to the client. The client may also include instructions in a request, but this behavior happens regardless of whether the client application needs to communicate with the server. If the server does not respond (as part of the application code) to the client request within the number of seconds set in this attribute, it does so anyway. The client accepts the response and automatically sends another request immediately.

Default is 45 seconds; valid range is 20 to 900 seconds.

`Tunneling Client Timeout`

If the number of seconds set in this attribute have elapsed since the client last sent a request to the server (in response to a reply), then the server regards the client as dead, and terminates the HTTP tunnel connection. The server checks the elapsed time at the interval specified by this attribute, when it would otherwise respond to the client's request.

Default is 40 seconds; valid range is 10 to 900 seconds.

Connecting to WebLogic Server from the Client

When your client requests a connection with WebLogic Server, all you need to do in order to use HTTP tunneling is specify the HTTP protocol in the URL. For example:

```
Hashtable env = new Hashtable();
env.put(Context.PROVIDER_URL, "http://wlhost:80");
Context ctx = new InitialContext(env);
```

On the client side, a special tag is appended to the `http` protocol, so that WebLogic Server knows this is a tunneling connection, instead of a regular HTTP request. Your application code does not need to do any extra work to make this happen.

The client must specify the port in the URL, even if the port is 80. You can set up your WebLogic Server to listen for HTTP requests on any port, although the most common choice is port 80 since requests to port 80 are customarily allowed through a firewall.

You specify the listen port for WebLogic Server in the Administration Console under the “Servers” node, under the “Network” tab.

Using Native I/O for Serving Static Files (Windows Only)

When running WebLogic Server on Windows NT/2000 you can specify that WebLogic Server use the native operating system call TransmitFile instead of using Java methods to serve static files such as HTML files, text files, and image files. Using native I/O can provide performance improvements when serving larger static files.

To use native I/O, add two parameters to the web.xml deployment descriptor of a Web Application containing the files to be served using native I/O. The first parameter, `weblogic.http.nativeIOEnabled` should be set to TRUE to enable native I/O file serving. The second parameter, `weblogic.http.minimumNativeFileSize` sets the minimum file size for using native I/O. If the file being served is larger than this value, native I/O is used. If you do not specify this parameter, a value of 400K is used.

Generally, native I/O provides greater performance gains when serving larger files; however, as the load on the machine running WebLogic Server increases, these gains diminish. You may need to experiment to find the correct value for `weblogic.http.minimumNativeFileSize`.

The following example shows the complete entries that should be added to the web.xml deployment descriptor. These entries must be placed in the web.xml file after the `<distributable>` element and before `<servlet>` element.

```
<context-param>
   <param-name>weblogic.http.nativeIOEnabled</param-name>
   <param-value>TRUE</param-value>
</context-param>
<context-param>
   <param-name>weblogic.http.minimumNativeFileSize</param-name>
   <param-value>500</param-value>
</context-param>
```

`weblogic.http.nativeIOEnabled` can also be set is as a context parameter in the FileServlet.

CHAPTER

7 Managing Transactions

These sections discuss transaction management and provide guidelines for configuring and managing transactions through the Administration Console.

For information on configuring JDBC connection pools to allow JDBC drivers to participate in distributed transactions, see "Managing JDBC Connectivity" on page 8-1.

Overview of Transaction Management

You use the Administration Console to access tools for configuring the WebLogic Server features, including the Java Transaction API (JTA). To invoke the Administration Console, see the procedures provided in Starting and Using the Administration Console in the *Administration Guide* at `http://e-docs.bea.com/wls/docs70/adminguide/overview.html#start_admin_console`. The transaction configuration process involves specifying values for attributes. These attributes define various aspects of the transaction environment:

- Transaction timeouts and limits
- Transaction Manager behavior
- Transaction log file prefix

Settings you make in the Administration Console, including configuration settings for JTA, are persisted in the `config.xml` file for the domain. For information about entries in this file, see the following sections of the *Configuration Reference Guide*:

- JTA at `http://e-docs.bea.com/wls/docs70/config_xml/JTA.html`
- JTAMigratableTarget at `http://e-docs.bea.com/wls/docs70/config_xml/JTAMigratableTarget.html`
- JTARecoveryService at `http://e-docs.bea.com/wls/docs70/config_xml/JTARecoveryService.html`
- JDBCTxDataSource at `http://e-docs.bea.com/wls/docs70/config_xml/JDBCTxDataSource.html`

Before configuring your transaction environment, you should be familiar with the J2EE components that can participate in transactions, such as EJBs, JDBC, and JMS.

- EJBs (Enterprise JavaBeans) use JTA for transaction support. Several deployment descriptors relate to transaction handling. For more information about programming with EJBs and JTA, see *Programming WebLogic Enterprise JavaBeans*.
- JDBC (Java Database Connectivity) provides standard interfaces for accessing relational database systems from Java. JTA provides transaction support on connections retrieved using a JDBC driver and transaction data source. For more information about programming with JDBC and JTA, see *Programming WebLogic JDBC*.
- JMS (Java Messaging Service) uses JTA to support transactions across multiple data resources. WebLogic JMS is an XA-compliant resource manager. For more

information about programming with JMS and JTA, see *Programming WebLogic JMS*.

For more information about configuring J2EE components, see the applicable sections of this document and the *Administration Console Online Help*.

Configuring Transactions

Configuration settings for JTA are applicable at the domain level. This means that configuration attribute settings apply to all servers within a domain. Monitoring and logging tasks for JTA are performed at the server level.

You can configure any transaction attributes before starting the server (static configuration) or, with one exception, while the server is running (dynamic configuration). The `TransactionLogFilePrefix` attribute must be set before starting the server.

To modify transaction attributes, do the following:

1. Start the Administration Console.
2. Select the domain node in the left pane. The Configuration tab for the domain is displayed by default.
3. Select the JTA tab.
4. For each attribute, change the value as desired or accept the default value.
5. Click Apply to store new attribute values.
6. Ensure that the `TransactionLogFilePrefix` attribute is set appropriately when you configure the server. For more information about setting the logging attribute, see "Monitoring and Logging Transactions" on page 7-10.

For detailed information about the transaction attributes available with WebLogic Server, including valid and default values, see the Domain topic in the Administration Console Online Help.

Additional Attributes for Managing Transactions

By default, if an XA resource that is participating in a global transaction fails to respond to an XA call from the WebLogic Server transaction manager, WebLogic Server flags the resource as unhealthy and unavailable, and blocks any further calls to the resource in an effort to preserve resource threads. The failure can be caused by either an unhealthy transaction or an unhealthy resource—there is no distinction between the two causes. In both cases, the resource is marked as unhealthy.

To mitigate this limitation, WebLogic Server provides the configuration attributes listed in Table 7-1:

Table 7-1 XA Resource Health Monitoring Configuration Attributes

Attribute	MBean	Definition
`EnableResourceHealthMonitoring`	`weblogic.management.configuration.JDBCConnectionPoolMBean`	Enables or disables resource health monitoring for the JDBC connection pool. This attribute only applies to connection pools that use an XA JDBC driver for database connections. It is ignored if a non-XA JDBC driver is used. If set to `true`, resource health monitoring is enabled. If an XA resource fails to respond to an XA call within the period specified in the `MaxXACallMillis` attribute, WebLogic Server marks the connection pool as unhealthy and blocks any further calls to the resource. If set to `false`, the feature is disabled. Default: `true`
`MaxXACallMillis`	`weblogic.management.configuration.JTAMBean`	Sets the maximum allowed duration (in milliseconds) of XA calls to XA resources. This setting applies to the entire domain. Default: `120000`

Table 7-1 XA Resource Health Monitoring Configuration Attributes

Attribute	MBean	Definition
`MaxResourceUnavailableMillis`	`weblogic.management.configuration.JTAMBean`	The maximum duration (in milliseconds) that an XA resource is marked as unhealthy. After this duration, the XA resource is declared available again, even if the resource is not explicitly re-registered with the transaction manager. This setting applies to the entire domain. Default: `1800000`
`MaxResourceRequestOnServer`	`weblogic.management.configuration.JTAMBean`	Maximum number of concurrent requests to resources allowed for each server in the domain. Default: `50` Minimum: `10` Maximum: `java.lang.Integer.MAX_VALUE`

You set these attributes directly in the config.xml file when the domain is inactive. These attributes are not available in the Administration Console. The following example shows an excerpt of a configuration file with these attributes:

```
...

   <JTA
    MaxUniqueNameStatistics="5"
    TimeoutSeconds="300"
    RecoveryThresholdMillis="150000"
    MaxResourceUnavailableMillis="900000"
    MaxResourceRequestOnServer="60"
    MaxXACallMillis="180000"
   />

 <JDBCConnectionPool
  Name="XAPool"
  Targets="myserver"
  DriverName="weblogic.qa.tests.transaction.
    testsupport.jdbc.XADataSource"
  InitialCapacity="1"
  MaxCapacity="10"
  CapacityIncrement="2"
  RefreshMinutes="5"
  TestTableName="dual"
  EnableResourceHealthMonitoring="true"
  Properties="user=scott;password=tiger;server=dbserver1"
 />

 <JDBCTxDataSource
  Name="XADataSource"
  Targets="myserver"
  JNDIName="weblogic.jdbc.XADS"
  PoolName="XAPool"
 />

...
```

Configuring Domains for Inter-Domain Transactions

For a transaction manager to manage distributed transactions, the transaction manager must be able to communicate with all participating servers to prepare and then commit or rollback the transactions. This applies to cases when your WebLogic domain acts as the transaction manager or a transaction participant (resource) in a distributed transaction. The following sections describe how to configure your domain to enable inter-domain transactions.

- "Limitations for Inter-Domain Transactions" on page 7-6
- "Inter-Domain Transactions for WebLogic Server Domains" on page 7-7
- "Using Security Interoperability Mode" on page 7-8

Limitations for Inter-Domain Transactions

Please note the following limitations for inter-domain transactions:

- You cannot manually resolve incomplete transactions on resources from a WebLogic Server domain from WebLogic Server version 7.0 or earlier.

- The domains and all participating resources must have unique names. That is, you cannot have a JDBC connection pool, a server, or a domain with the same name as an object in another domain or the domain itself.
- Only one data source with *both* of the following attribute conditions can participate in a global transaction, regardless the domain in which the data source is configured:
  - Emulate Two-Phase Commit is selected (`EnableTwoPhaseCommit=true` in `config.xml`).
  - The Pool Name attribute in the data source specifies a connection pool that uses a non-XA driver to create database connections.

Note: BEA recommends that you use an XA driver instead of a non-XA driver (with Emulate Two-Phase Commit) in global transactions. There are risks involved with using a non-XA driver in a global transaction. See Limitations and Risks When Using a Non-XA Driver in Global Transactions in the *Administration Guide* for more details about the risks.

Inter-Domain Transactions for WebLogic Server Domains

To manage or participate in transactions that span multiple WebLogic Server domains (that is, all participating domains run on WebLogic Server 9.x, 8.x, 7.x, and 6.x domains or a combination of 9.x, 8.x, 7.x and 6.x), you must enable inter-domain transactions by establishing domain trust and setting the `Security Interoperability Mode`. For all participating domains:

1. In all participating WebLogic Server 6.x domains, change the password for the `system` user to the same value in all participating domains on the Security→Users tab in the Administration Console. See Changing the System Password at `http://e-docs.bea.com/wls/docs61/adminguide/cnfgsec.html#cnfgsec003`.
2. Establish domain trust by setting a security credential for all domains to the same value in all participating domains. If you have participating 6.x domains, set the security credential for all domains to the same value as the `system` password in all participating WebLogic Server 6.x domains.
   - For 7.x domains, see Enabling Trust Between WebLogic Domains in *Managing WebLogic Security.*

- For 8.x domains, see Enabling Trust Between WebLogic Domains in *Managing WebLogic Security*.
- For 9.x domains, see Enable trust between domains in *Administration Console Online Help*.

3. For each server participating in the transaction, set the `Security Interoperability Mode` flag according to Table 7-2 before rebooting. See "Using Security Interoperability Mode" on page 7-8.

Using Security Interoperability Mode

`Security Interoperability Mode` enables you to configure compatible communication channels between servers in global transactions.

- "Configuring Security Interoperability Mode" on page 7-8
- "Determining the Security Interoperability Mode" on page 7-9

Configuring Security Interoperability Mode

To configure `Security Interoperability Mode`, every participating server must set the following flag to the same value:

```
-Dweblogic.transaction.SecurityInteropMode=value
```

Where *value* is:

- **performance**—The transaction coordinator makes calls using `anonymous` at all times. This implies a security risk since a malicious third party could then try to affect the outcome of transactions using a man-in-the-middle attack.
- **compatibility**—The transaction coordinator makes calls as the kernel identity over an unsecure channel. This is a high security risk because a successful man-in-the-middle attack would allow the attacker to gain administrative control over both domains. This setting should only be used when strong network security is in place.

The default value of this flag is *`performance`*.

Determining the Security Interoperability Mode

Use the following table to determine the `Security Interoperability Mode` settings required when configuring communication channels for inter-domain transactions.

Table 7-2 Security Interoperability Settings for Inter-Domain Transactions

Domain	9.x	8.1 SP5 and higher	7.0 SP7 and higher	8.1 SP4 and lower	7.0 SP6 and lower	6.x
9.x	Set both domains to `default` or `performance`	Set both domains to `performance`	Set both domains to `performance`	Set the 9.x domain to `compatibility`	Set the 9.x domain to `compatibility`	Set the 9.x domain to `compatibility`
8.1 SP5 and higher	Set both domains to `performance`	Set both domains to `performance`	Set both domains to `performance`	Set the 8.1 SP5 and higher domain to `compatibility`	Set the 8.1 SP5 and higher domain to `compatibility`	Set the 8.1 SP5 and higher domain to `compatibility`
7.0 SP7 and higher	Set both domains to `performance`	Set both domains to `performance`	Set both domains to `performance`	Set the 7.0 SP7 and higher domain to `compatibility`	Set the 7.0 SP7 and higher domain to `compatibility`	Set the 7.0 SP7 and higher domain to `compatibility`
8.1 SP4 and lower	Set the 9.x domain to `compatibility`	Set the 8.1 SP5 and higher domain to `compatibility`	Set the 7.0 SP7 and higher domain to `compatibility`	N/A	N/A	N/A

Table 7-2 Security Interoperability Settings for Inter-Domain Transactions

Domain	9.x	8.1 SP5 and higher	7.0 SP7 and higher	8.1 SP4 and lower	7.0 SP6 and lower	6.x
7.0 SP6 and lower	Set the 9.x domain to `compatibility`	Set the 8.1 SP5 and higher domain to `compatibility`	Set the 7.0 SP7 and higher domain to `compatibility`	N/A	N/A	N/A
6.x	Set the 9.x domain to `compatibility`	Set the 8.1 SP5 and higher domain to `compatibility`	Set the 7.0 SP7 and higher domain to `compatibility`	N/A	N/A	N/A

Note: When `Security Interoperability Mode` is set to performance, you are not required to set domain trust between the domains.

Monitoring and Logging Transactions

The Administration Console allows you to monitor transactions and to specify the transaction log file prefix. Monitoring and logging tasks are performed at the server level. Transaction statistics are displayed for a specific server and each server has a transaction log file.

To display transaction statistics and to set the prefix for the transaction log files, do the following:

1. Start the Administration Console.
2. Click the server node in the left pane.
3. Select a specific server in the left pane.
4. Select the Monitoring tab.

5. Select the JTA tab. Totals for transaction statistics are displayed in the JTA dialog. (You can also click the monitoring text links to monitor transactions by resource or by name, or to monitor all active transactions.)
6. Select the Logging tab.
7. Select the JTA tab.
8. Enter a transaction log file prefix (storage location for transaction logs) then click Apply to save the attribute setting. The new transaction log file prefix takes effect after you restart the server.

   The default transaction log file prefix is the server's working directory.

For detailed information on monitoring and logging values and attributes, see the following sections in the *Administration Console Online Help*:

- Server →Monitoring →JTA at `http://e-docs.bea.com/wls/docs70/ConsoleHelp/domain_server_monitor_jta.html`
- Server →Logging →JTA at `http://e-docs.bea.com/wls/docs70/ConsoleHelp/domain_server_logging_jta.html`
- Domain →Configuration →JTA at `http://e-docs.bea.com/wls/docs70/ConsoleHelp/domain_domain_config_jta.html`

Transaction Monitoring

You can monitor transactions in progress using the WebLogic Console. In addition to displaying statistics, as described in Transaction Statistics in *Programming WebLogic JTA*, you can display the following:

- transactions by name, including rollback and time active information
- transactions by resource, including statistics on total, committed, and rolled back transactions
- all active transactions, including information on status, servers, resources, properties, and the transaction identifier

Transaction Log Files

Each server has a transaction log which stores information about committed transactions coordinated by the server that may not have been completed. WebLogic Server uses the transaction log when recovering from system crashes or network failures. You cannot directly view the transaction log—the file is in a binary format.

The transaction log consists of multiple files. Each file is subject to garbage collection. That is, when none of the records in a transaction log file are needed, the system deletes the file and returns the disk space to the file system. In addition, the system creates a new transaction log file if the previous log file becomes too large or a checkpoint occurs.

Caution: Do not manually delete transaction log files. Deleting transaction log files may cause inconsistencies in your data.

Transaction log files are uniquely named using a pathname prefix, the server name, a four-digit numeric suffix, and a file extension. The pathname prefix determines the storage location for the file. You can specify a value for the `TransactionLogFilePrefix` server attribute using the WebLogic Administration Console. The default `TransactionLogFilePrefix` is the server's working directory.

You should set the `TransactionLogFilePrefix` so that transaction log files are created on a highly available file system, for example, on a RAID device. To take advantage of the migration capability of the Transaction Recovery Service for servers in a cluster, you must store the transaction log in a location that is available to a server and its backup servers, preferably on a dual-ported SCSI disk or on a Storage Area Network (SAN). See "Preparing to Migrate the Transaction Recovery Service" on page 7-23 for more information.

On a UNIX system with a server name of `websvr` and with the `TransactionLogFilePrefix` set to `/usr7/applog1/`, you might see the following log files:

```
/usr7/applog1/websvr0000.tlog
/usr7/applog1/websvr0001.tlog
/usr7/applog1/websvr0002.tlog
```

Similarly, on a Windows system with the `TransactionLogFilePrefix` set to `C:\weblogic\logA\`, you might see the following log files:

```
C:\weblogic\logA\websvr0000.tlog
C:\weblogic\logA\websvr0001.tlog
C:\weblogic\logA\websvr0002.tlog
```

If you notice a large number of transaction log files on your system, this may be an indication of multiple long-running transactions that have not completed. This can be caused by resource manager failures or transactions with especially large timeout values.

If the file system containing the transaction log runs out of space or is inaccessible, `commit()` throws `SystemException`, and the transaction manager places a message in the system error log. No transactions are committed until more space is available.

When migrating a server to another machine, move the transaction log files as well, keeping all the log files for a server together. See "Moving a Server to Another Machine" on page 7-17 for more information.

Setting the Transaction Log File Write Policy

You can select a transaction log file write policy to change the way WebLogic Server writes transaction log file entries. You can select either of the following options:

- Cache-Flush—(the default) Flushes operating system and on-disk caches after each entry to the transaction log. Transactions cannot commit until the commit record is written to stable storage.
- Direct-Write—Forces the operating system to write transaction log entries directly to disk with each write. This option is available on Windows, Solaris and HP-UX platforms.

Warning: On Windows, the Direct-Write transaction log file write policy may leave transaction data in the on-disk cache without immediately writing it to disk. This is not transactionally safe because a power failure can cause loss of on-disk cache data. To prevent cache data loss when using the Direct-Write transaction log file write policy on Windows, disable all write caching for the disk (enabled by default) or use a battery backup for the system.

The transaction log file write policy can affect transaction performance. You should test these options with your system to see which performs better. Direct-Write typically performs as well or better than Cache-Flush, depending on operating system and OS parameter settings, and is available on Windows, HP-UX, and Solaris. Windows systems optimize serial writes to disk such that subsequent writes to a file get faster after the first write to the file. Transaction log file entries are written serially, so this could improve performance. On some UNIX systems, the Cache-Flush option will flush all cached disk writes, not only those for the transaction log file, which could degrade transaction performance.

To set the transaction log file write policy, follow these steps:

1. In the Administration Console, click the server node in the left pane and select a server.
2. In the right pane, select the Logging tab and then the JTA tab.
3. Select a Transaction Log File Write Policy: Cache-Flush (the default) or Direct Write.
4. Click Apply to save the attribute setting. The new transaction log file write policy takes effect after you restart the server.

Heuristic Log Files

When importing transactions from a foreign transaction manager into WebLogic Server, the WebLogic Server transaction manager acts as an XA resource coordinated by the foreign transaction manager. In rare catastrophic situations, such as after the transaction abandon timeout expires or if the XA resources participating in the WebLogic Server imported transaction throw heuristic exceptions, the WebLogic Server transaction manager will make a heuristic decision. That is, the WebLogic Server transaction manager will decide to commit or roll back the transaction without input from the foreign transaction manager. If the WebLogic Server transaction manager makes a heuristic decision, it stores the information of the heuristic decision in the heuristic log files until the foreign transaction manager tells it to forget the transaction.

Heuristic log files are stored with transaction log files and look similar to transaction log files with `.heur` before the `.tlog` extension. They use the following format:

```
<TLOG_file_prefix>\<server_name><4-digit number>.heur.tlog
```

On a UNIX system with a server name of `websvr`, you might see the following heuristic log files:

```
/usr7/applog1/websvr0000.heur.tlog
/usr7/applog1/websvr0001.heur.tlog
/usr7/applog1/websvr0002.heur.tlog
```

Similarly, on a Windows system, you might see the following heuristic log files:

```
C:\weblogic\logA\websvr0000.heur.tlog
C:\weblogic\logA\websvr0001.heur.tlog
C:\weblogic\logA\websvr0002.heur.tlog
```

Handling Heuristic Completions

An **heuristic completion** (or heuristic decision) occurs when a resource makes a unilateral decision during the completion stage of a distributed transaction to commit or rollback updates. This can leave distributed data in an indeterminate state. Network failures or resource timeouts are possible causes for heuristic completion. In the event of an heuristic completion, one of the following heuristic outcome exceptions may be thrown:

- `HeuristicRollback`—one resource participating in a transaction decided to autonomously rollback its work, even though it agreed to prepare itself and wait for a commit decision. If the Transaction Manager decided to commit the transaction, the resource's heuristic rollback decision was incorrect, and might lead to an inconsistent outcome since other branches of the transaction were committed.
- `HeuristicCommit`—one resource participating in a transaction decided to autonomously commit its work, even though it agreed to prepare itself and wait for a commit decision. If the Transaction Manager decided to rollback the transaction, the resource's heuristic commit decision was incorrect, and might lead to an inconsistent outcome since other branches of the transaction were rolled back.
- `HeuristicMixed`—the Transaction Manager is aware that a transaction resulted in a mixed outcome, where some participating resources committed and some rolled back. The underlying cause was most likely heuristic rollback or heuristic commit decisions made by one or more of the participating resources.
- `HeuristicHazard`—the Transaction Manager is aware that a transaction might have resulted in a mixed outcome, where some participating resources

committed and some rolled back. But system or resource failures make it impossible to know for sure whether a Heuristic Mixed outcome definitely occurred. The underlying cause was most likely heuristic rollback or heuristic commit decisions made by one or more of the participating resources.

When an heuristic completion occurs, a message is written to the server log. Refer to your database vendor documentation for instructions on resolving heuristic completions.

Some resource managers save context information for heuristic completions. This information can be helpful in resolving resource manager data inconsistencies. If the `ForgetHeuristics` attribute is selected (set to true) on the JTA panel of the WebLogic Console, this information is removed after an heuristic completion. When using a resource manager that saves context information, you may want to set the `ForgetHeuristics` attribute to false.

Moving a Server

A server instance is identified by its URL (IP address or DNS name plus the listening port number). Changing the URL by moving the server to a new machine or changing the Listening Port of a server on the same machine effectively moves the server so the server identity may no longer match the information stored in the transaction logs.

- If the new server has the same URL as the old server, the Transaction Recovery Service searches all transaction log files for incomplete transactions and completes them as described in "Transaction Recovery Service Actions After a Crash" on page 7-19.
- If the new server does not have the same URL, any pending transactions stored in the transaction log files are unrecoverable. If you wish, you can delete the transaction log files. This step prevents the Transaction Recovery Service from attempting to resolve these transactions until the value of the `AbandonTimeoutSeconds` parameter is exceeded. See "Abandoning Transactions" on page 7-17 for more information.
- If a server acting as a remote transaction sub-coordinator fails and its URL changes, any ongoing transactions will not complete (commit or rolledback) because the coordinator is unable to communicate with the remote sub-coordinator. The coordinator will attempt the commit or rollback request

until `AbandonTimeoutSeconds` is exceeded. See “Abandoning Transactions” on page 7-17 for more information.

BEA recommends configuring server instances using DNS names rather than IP addresses to promote portability.

Moving a Server to Another Machine

Note: BEA recommends moving the transaction log files to the new machine before starting the server on the new machine. By doing so, you ensure that the transaction recovery process runs properly. When you start WebLogic Server on the new system, the server reads the transaction log files to recover pending transactions, if any.

When an application server is moved to another machine, it must be able to locate the transaction log files on the new disk. If the pathname is different on the new machine, update the `TransactionLogFilePrefix` attribute with the new path before starting the server. For instructions on how to change the `TransactionLogFilePrefix`, see Specifying the Transaction Log File Location in the *Administration Console Online Help*.

Abandoning Transactions

You can choose to abandon incomplete transactions after a specified amount of time. In the two-phase commit process for distributed transactions, the transaction manager coordinates all resource managers involved in a transaction. After all resource managers vote to commit or rollback, the transaction manager notifies the resource managers to act—to either commit or rollback changes. During this second phase of the two-phase commit process, the transaction manager will continue to try to complete the transaction until all resource managers indicate that the transaction is completed. Using the `AbandonTimeoutSeconds` attribute, you can set the maximum time, in seconds, that a transaction manager will persist in attempting to complete a transaction during the second phase of the commit protocol. The default value is 86400 seconds, or 24 hours. After the abandon transaction timer expires, no further attempt is made to resolve the transaction with any resources that are unavailable or unable to acknowledge the transaction outcome. If the transaction is in a prepared state before being abandoned, the transaction manager will roll back the transaction to release any locks held on behalf of the abandoned transaction and will write an heuristic error to the server log. For more information, see:

- For instructions on how to set the `AbandonTimeoutSeconds` attribute, see Configuring JTA in the *Administration Console Online Help*.
- For more information about the two-phase commit process, see Distributed Transactions and the Two-Phase Commit Protocol in *Programming WebLogic JTA*.

Transaction Recovery After a Server Fails

The WebLogic Server transaction manager is designed to recover from system crashes with minimal user intervention. The transaction manager makes every effort to resolve transaction branches that are prepared by resource managers with a commit or roll back, even after multiple crashes or crashes during recovery.

To facilitate recovery after a crash, WebLogic Server provides the Transaction Recovery Service, which automatically attempts to recover transactions on system startup. The Transaction Recovery Service owns the transaction log for a server. On startup, the Transaction Recovery Service parses all log files for incomplete transactions and completes them as described in "Transaction Recovery Service Actions After a Crash" on page 7-19.

Because the Transaction Recovery Service is designed to gracefully handle transaction recovery after a crash, BEA recommends that you attempt to restart a crashed server and allow the Transaction Recovery Service to handle incomplete transactions.

If a server crashes and you do not expect to be able to restart it within a reasonable period of time, you may need to take action. Procedures for recovering transactions after a server failure differ based on your WebLogic Server environment. For a non-clustered server, you can manually move the server (with transaction log files) to another system (machine) to recover transactions. See "Recovering Transactions for a Failed Non-Clustered Server" on page 7-20 for more information. For a server in a cluster, you can manually *migrate* the *Transaction Recovery Service* to another server in the same cluster. Migrating the Transaction Recovery Service involves selecting a server with access to the transaction logs to recover transactions, and then migrating the service using the Administration Console or the WebLogic command line interface.

Note: For non-cluster servers, you can only move the entire server to a new system. For clustered servers, you can temporarily migrate the Transaction Recovery Service.

For more information about migrating the Transaction Recovery Service, see "Recovering Transactions for a Failed Clustered Server" on page 7-21. For more information about clusters, see *Using WebLogic Server Clusters* at http://e-docs.bea.com/wls/docs70/cluster/index.html.

Transaction Recovery Service Actions After a Crash

When you restart a server after a crash or when you migrate the Transaction Recovery Service to another (backup) server, the Transaction Recovery Service does the following:

- Complete transactions ready for second phase of two-phase commit

  For transactions for which a commit decision has been made but the second phase of the two-phase commit process has not completed (transactions recorded in the transaction log), the Transaction Recovery Service completes the commit process.

- Resolve prepared transactions

  For transactions that the transaction manager has prepared with a resource manager (transactions in phase one of the two-phase commit process), the Transaction Recovery Service must call `XAResource.recover()` during crash recovery for each resource manager and eventually resolve (by calling the `commit()`, `rollback()`, or `forget()` method) all transaction IDs returned by `recover()`.

- Report heuristic completions

  If a resource manager reports a heuristic exception, the Transaction Recovery Service records the heuristic exception in the server log and calls `forget()` if the `Forget Heuristics` configuration attribute is enabled. If the `Forget Heuristics` configuration attribute is not enabled, refer to your database vendor's documentation for information about resolving heuristic completions. See "Handling Heuristic Completions" on page 7-15 for more information.

The Transaction Recovery Service provides the following benefits:

- Maintains consistency across resources

  The Transaction Recovery Service handles transaction recovery in a consistent, predictable manner: For a transaction for which a commit decision has been made but is not yet committed before a crash, and `XAResource.recover()` returns the transaction ID, the Transaction Recovery Service consistently calls `XAResource.commit()`; for a transaction for which a commit decision has not been made before a crash, and `XAResource.recover()` returns its transaction ID, the Transaction Recovery Service consistently calls `XAResource.rollback()`. With consistent, predictable transaction recovery, a transaction manager crash by itself cannot cause a mixed heuristic completion where some branches are committed and some are rolled back.

- Persists in achieving transaction resolution

  If a resource manager crashes, the Transaction Recovery Service must eventually call `commit()` or `rollback()` for each prepared transaction until it gets a successful return from `commit()` or `rollback()`. The attempts to resolve the transaction can be limited by setting the `AbandonTimeoutSeconds` configuration attribute. See "For instructions on how to set the `AbandonTimeoutSeconds` attribute, see Configuring JTA in the Administration Console Online Help. For more information about the two-phase commit process, see Distributed Transactions and the Two-Phase Commit Protocol in Programming WebLogic JTA." on page 7-19 for more information.

Recovering Transactions for a Failed Non-Clustered Server

To recover transactions for a failed server, follow these steps:

1. Move (or make available) all transaction log files from the failed server to a new server.

2. Set the `TransactionLogFilePrefix` attribute with the path to the transaction log files. For instructions, see Specifying the Transaction Log File Location in the *Administration Console Online Help*.

3. Start the new server. The Transaction Recovery Service searches all transaction log files for incomplete transactions and completes them as described in "Transaction Recovery Service Actions After a Crash" on page 7-19.

When moving transaction logs after a server failure, make all transaction log files available on the new machine before starting the server there. You can accomplish this by storing transaction log files on a dual-ported disk available to both machines. As in the case of a planned migration, update the `TransactionLogFilePrefix` attribute with the new path before starting the server if the pathname is different on the new machine. Ensure that all transaction log files are available on the new machine before the server is started there. Otherwise, transactions in the process of being committed at the time of a crash might not be resolved correctly, resulting in application data inconsistencies.

Note: The Transaction Recovery Service is designed to gracefully handle transaction recovery after a crash. BEA recommends that you attempt to restart a crashed server and allow the Transaction Recovery Service to handle incomplete transactions, rather than move the server to a new machine.

Recovering Transactions for a Failed Clustered Server

When a clustered server crashes, you can manually migrate the Transaction Recovery Service from the crashed server to another server in the same cluster using the Administration Console or the command line interface. The following events occur:

1. The Transaction Recovery Service on the backup server takes ownership of the transaction log from the crashed server.

2. The Transaction Recovery Service searches all transaction log files from the failed server for incomplete transactions and completes them as described in "Transaction Recovery Service Actions After a Crash" on page 7-19.

3. If the Transaction Recovery Service on the backup server successfully completes all incomplete transactions from the failed server, the server releases ownership of the Transaction Recovery Service (including transaction log files) for the failed server so the failed server can reclaim it upon restart.

For instructions to migrate the Transaction Recovery Service using the Administration Console, see Migrating the Transaction Recovery Service to a Server in the Same Cluster in the Administration Console online help at `http://e-docs.bea.com/wls/docs70/ConsoleHelp/jta.html#jta_trs_migrate`. For instructions to migrate the Transaction Recovery Service using the command line interface, see MIGRATE in Appendix B, "WebLogic Server Command-Line Interface Reference."

A server can perform transaction recovery for more than one failed server. While recovering transactions for other servers, the backup server continues to process and recover its own transactions. If the backup server fails during recovery, you can migrate the Transaction Recovery Service to yet another server, which will continue the transaction recovery. You can also manually migrate the Transaction Recovery Service back to the original failed server using the Administration Console or the command line interface. See Manually Migrating the Transaction Recovery Service Back to the Original Server in the Administration Console online help at `http://e-docs.bea.com/wls/docs70/ConsoleHelp/jta.html#jta_trs_migrate_back` for more information.

When a backup server completes transaction recovery for a server, it releases ownership of the Transaction Recovery Service (and transaction logs) for the failed server. When you restart a failed server, it attempts to reclaim ownership of its Transaction Recovery Service. If a backup server is in the process of recovering transactions when you restart the failed server, the backup server stops recovering transactions, performs some internal cleanup, and releases ownership of the Transaction Recovery service so the failed server can reclaim it and start properly. The failed server will then complete its own transaction recovery.

If a backup server still owns the Transaction Recovery Service for a failed server and the backup server is inactive when you attempt to restart the failed server, the failed server will not start because the backup server cannot release ownership of the Transaction Recovery Service. This is also true if the fail back mechanism fails or if the backup server cannot communicate with the Administration Server. You can manually migrate the Transaction Recovery using the Administration Console or the command line interface.

Limitations of Migrating the Transaction Recovery Service

When migrating the Transaction Recovery Service, the following limitations apply:

- You cannot migrate the Transaction Recovery Service to a backup server from a server that is running. You must stop the server before migrating the Transactions Recovery Service.
- The backup server does not accept new transaction work for the failed server. It only processes incomplete transactions.
- The backup server does not process heuristic log files.
- The backup server only processes log records written by WebLogic Server. It does not process log records written by gateway implementations, including WebLogic Tuxedo Connector.

Preparing to Migrate the Transaction Recovery Service

To migrate the Transaction Recovery Service from a failed server in a cluster to another server (backup server) in the same cluster, the backup server must have access to the transaction log files from the failed server. Therefore, you must store transaction log files on persistent storage available to both (or more) servers. BEA recommends that you store transaction log files on a Storage Area Network (SAN) device or a dual-ported disk. Do not use an NFS file system to store transaction log files. Because of the caching scheme in NFS, transaction log files on disk may not always be current. Using transaction log files stored on an NFS device for recovery may cause data corruption.

When migrating the Transaction Recovery Service from a server, you must stop the failing or failed server before actually migrating the Transaction Recovery Service. If the original server is still running, you cannot migrate the Transaction Recovery Service from it.

For detailed instructions to migrate the Transaction Recovery Service, see Migrating the Transaction Recovery Service to a Server in the Same Cluster in the *Administration Console Online Help* at `http://e-docs.bea.com/wls/docs70/ConsoleHelp/jta.html#jta_trs_migrate`.

You can also use the command line to migrate the Transaction Recovery Service. See "MIGRATE" in Appendix B, "WebLogic Server Command-Line Interface Reference."

CHAPTER

8 Managing JDBC Connectivity

The following sections provide guidelines for configuring and managing database connectivity through the JDBC components—Data Sources, connection pools and MultiPools—for both local and distributed transactions:

Overview of JDBC Administration

The Administration Console provides an interface to the tools that allow you to configure and manage WebLogic Server features, including JDBC (Java database connectivity). For most JDBC administrative functions, which include creating, managing and monitoring connectivity, systems administrators use the Administration Console or the command-line interface. Application developers may want to use the JDBC API or the WebLogic Management API.

Frequently performed tasks to set and manage connectivity include:

- Defining the attributes that govern JDBC connectivity between WebLogic Server and your database management system
- Managing established connectivity
- Monitoring established connectivity

About the Administration Console

Your primary way to set and manage JDBC connectivity is through the Administration Console. Using the Administration Console, you set up persistent connectivity—connection pools, Data Sources, Tx Data Sources, and Multipools that are available even after you stop and restart the server. These JDBC objects are known as *static* objects. You can create *dynamic* objects—objects that you expect to use and then remove—with the administration command line or in application code.

In addition to setting connectivity, the Administration Console allows you to manage and monitor established connectivity.

About the Command-Line Interface

The command-line interface provides a way to dynamically create and manage connection pools and data sources. For information on how to use the command-line interface, see "WebLogic Server Command-Line Interface Reference" on page B-1.

About the JDBC API

For information on setting and managing connectivity programmatically, see Programming WebLogic JDBC at `http://e-docs.bea.com/wls/docs70/jdbc/index.html`.

Related Information

The JDBC drivers, used locally and in distributed transactions, interface with many WebLogic Server components and information appears in several documents. For example, information about JDBC drivers is included in the documentation sets for JDBC, JTA, and WebLogic jDrivers.

Here is a list of additional resources for JDBC, JTA and Administration:

Administration and Management

- For instructions on opening the Administration Console, refer to "Starting and Using the Administration Console" on page 1-22.
- For a complete list of the JDBC attributes, see the *WebLogic Server Configuration Reference Guide* at `http://e-docs.bea.com/wls/docs70/config_xml/index.html`.
- For information about using the command-line interface, see "WebLogic Server Command-Line Interface Reference" on page B-1.

JDBC and WebLogic jDrivers

The following documentation is written primarily for application developers. Systems Administrators may want to read the introductory material as a supplement to the material in this document.

- For information about the JDBC API, see *Programming WebLogic JDBC*. The "Introduction to WebLogic JDBC" section provides a concise overview of JDBC and JDBC drivers.
- For information about using the WebLogic jDrivers, see *Using WebLogic jDriver for Oracle* at http://e-docs.bea.com/wls/docs70/oracle/index.html or *Using WebLogic jDriver for Microsoft SQL Server* at http://e-docs.bea.com/wls/docs70/mssqlserver4/index.html.

Transactions (JTA)

- For information about managing JTA, see "Managing Transactions" on page 7-1.

- For information about using third-party drivers, see "Using Third-Party JDBC XA Drivers with WebLogic Server" in *Programming WebLogic JTA* at http://e-docs.bea.com/wls/docs70/jta/thirdpartytx.html.

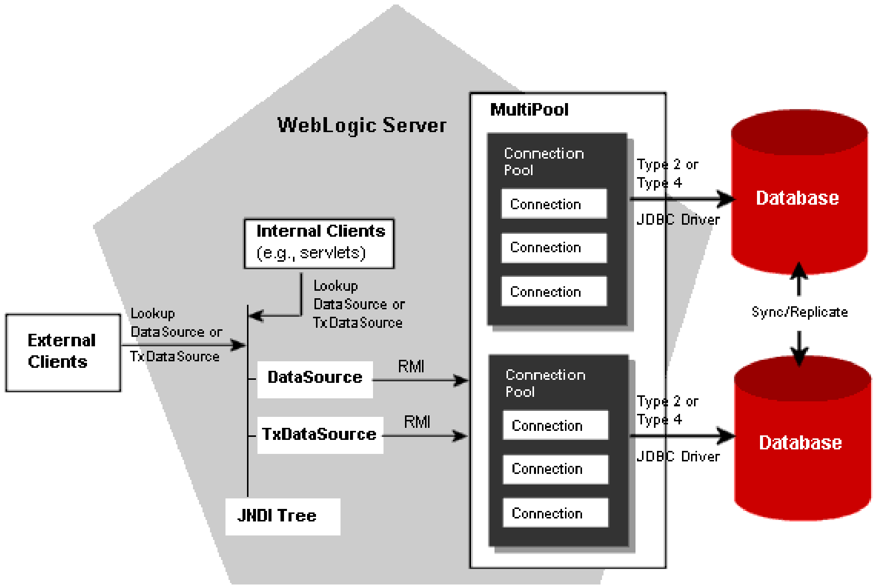
The following documentation is written primarily for application developers. Systems Administrators may want to read the following as supplements to the material in this section.

- For information about distributed transactions, see *Programming WebLogic JTA* at http://e-docs.bea.com/wls/docs70/jta/index.html.
- For information about using the WebLogic jDriver for Oracle/XA, see "Using WebLogic jDriver for Oracle/XA in Distributed Transactions" in *Using WebLogic jDriver for Oracle* at http://e-docs.bea.com/wls/docs70/oracle/trxjdbcx.html.

JDBC Components—Connection Pools, Data Sources, and MultiPools

The following sections provide a brief overview of the JDBC connectivity components—connection pools, MultiPools, and Data Sources.

Figure 8-1 JDBC Components in WebLogic Server



Connection Pools

A connection pool contains a group of JDBC connections that are created when the connection pool is registered—when starting up WebLogic Server or when assigning the connection pool to a target server or cluster. Connection pools use a type 2 or type 4 JDBC driver to create physical database connections. Your application borrows a connection from the pool, uses it, then returns it to the pool by closing it. Read more about Connection Pools in *Programming WebLogic JDBC* at `http://e-docs.bea.com/wls/docs70/jdbc/programming.html`.

All of the settings you make with the Administration Console are static; that is, all settings persist even after you stop and restart WebLogic Server. You can create dynamic connection pools—those that you expect to use and delete while the server is running—using the command line (see “WebLogic Server Command-Line Interface Reference” on page B-1) or programmatically using the API (see Creating a

Connection Pool Dynamically in *Programming WebLogic JDBC* at `http://e-docs.bea.com/wls/docs70/jdbc/programming.html#dynamic_conn_pool`).

Connection pool settings are persisted in the `config.xml` file, including settings for dynamically created connection pools (until you programmatically delete the connection pool). For information about entries in the `config.xml` file, see the JDBCConnectionPool section of the *Configuration Reference Guide* at `http://e-docs.bea.com/wls/docs70/config_xml/JDBCConnectionPool.html`.

Application-Scoped JDBC Connection Pools

When you package your enterprise applications, you can include the `weblogic-application.xml` supplemental deployment descriptor, which you use to configure *application scoping*. Within the `weblogic-application.xml` file, you can configure JDBC connection pools that are created when you deploy the enterprise application.

An instance of the connection pool is created with each instance of your application. This means an instance of the pool is created with the application on each node that the application is targeted to. It is important to keep this in mind when considering pool sizing.

Connection pools created in this manner are known as *application-scoped connection pools*, *app scoped pools*, *application local pools*, *app local pools*, or *local pools*, and are scoped for the enterprise application only. That is, they are isolated for use by the enterprise application.

For more information about application scoping and application scoped resources, see:

- weblogic-application.xml Deployment Descriptor Elements in *Developing WebLogic Server Applications* at `http://e-docs.bea.com/wls/docs70/programming/app_xml.html#app-scoped-pool`.
- Packaging Enterprise Applications in *Developing WebLogic Server Applications* at `http://e-docs.bea.com/wls/docs70/programming/packaging.html#pack009`.

- Two-Phase Deployment in *Developing WebLogic Server Applications* at `http://e-docs.bea.com/wls/docs70/programming/deploying.html#twophasedeploy`.

MultiPools

A MultiPool is a pool of connection pools. Used in local (non-distributed) transactions on single or multiple WebLogic Server configurations, MultiPools aid in either:

- Load Balancing—pools are accessed using a round-robin scheme. When switching connections, WebLogic Server selects a connection from the next connection pool in the order listed.
- High Availability—connection pools are listed in the order that determines the order in which connection pool switching occurs. That is, WebLogic Server provides database connections from the first connection pool on the list. If that connection pool fails, it attempts to use a database connection from the second, and so forth.

Data Sources

A Data Source object enables JDBC applications to obtain a DBMS connection from a connection pool. Each Data Source object binds to the JNDI tree and points to a connection pool or MultiPool. Applications look up the Data Source to get a connection. Data Source objects can be defined with JTA (Tx Data Sources in the Administration Console) or without JTA (Data Sources in the Administration Console). You use Tx Data Source for distributed transactions. See "JDBC Configuration Guidelines for Connection Pools, MultiPools, and DataSources" on page 8-19 for more information about using Data Sources and Tx Data Sources.

JDBC Data Source Factories

In WebLogic Server, you can bind a JDBC DataSource resource into the WebLogic Server JNDI tree as a resource factory. You can then map a resource factory reference in the EJB deployment descriptor to an available resource factory in a running WebLogic Server to get a connection from a connection pool.

For details about creating and using a JDBC Data Source factory, see Resource Factories in *Programming WebLogic Enterprise JavaBeans* at `http://e-docs.bea.com/wls/docs70/ejb/EJB_environment.html#resourcefact`.

Security for JDBC Connection Pools

You can optionally restrict access to JDBC connection pools. In previous releases of WebLogic Server, ACLs were used to protect WebLogic resources. In WebLogic Server version 7.0, security policies answer the question "who has access" to a WebLogic resource. A security policy is created when you define an association between a WebLogic resource and a user, group, or role. A WebLogic resource has no protection until you assign it a security policy. For instructions on how to set up security for all WebLogic Server resources, see "Setting Protections for WebLogic Resources" in the *Administration Console Online Help*.

Security for JDBC Connection Pools in Compatibility Mode

WebLogic Server 7.0 continues to support the security model from version 6.1 for backward compatibility. To use version 6.1 security, you must run in compatibility mode. For details about running in compatibility mode, see the following documents:

- Security Compatibility Mode in the *Administration Console Online Help* at `http://e-docs.bea.com/wls/docs70/ConsoleHelp/security_6x.html`.
- Deploying a WebLogic Server 6.x Application on WebLogic Server 7.0 in the *Upgrade Guide* at `http://e-docs.bea.com/wls/docs70/upgrade/upgrade6xto70.html#deploying`.
- Security in the *Upgrade Guide* at `http://e-docs.bea.com/wls/docs70/upgrade/upgrade6xto70.html#security`.

The default security realm for WebLogic Server 6.1 was the File realm, which uses ACLs in the `fileRealm.properties` file for authorization and authentication. Connection pools are unprotected unless you define ACLs for connection pools (as a resource type) or for individual connection pools. If you define an ACL for connection pools, access is restricted to *exactly* what is defined in the ACL. For example, before you have any ACLs for connection pools in your `fileRealm.properties` file, everyone has unrestricted access to all connection pools in your domain. However, if you add the following line to the file, access becomes very restricted:

```
acl.reset.weblogic.jdbc.connectionPool=Administrators
```

This line grants reset privileges to Administrators on all connection pools *and it prohibits all other actions by all other users*. By adding an ACL, file realm protection for connection pools is activated. WebLogic Server enforces the ACLs defined in `fileRealm.properties` and only allows access specifically granted in the file. If your intent in adding the ACL was to restrict resets only on connection pools, you must specifically grant privileges for other actions to everyone or to specific roles or users. For example:

```
acl.reserve.weblogic.jdbc.connectionPool=everyone
acl.shrink.weblogic.jdbc.connectionPool=everyone
acl.admin.weblogic.jdbc.connectionPool=everyone
```

Table 8-1 lists the ACLs that you can use in `fileRealm.properties` to secure connection pools.

Table 8-1 File Realm JDBC ACLs

Use this ACL . . .	To Restrict . . .
`reserve.weblogic.jdbc.connectionPool[.`*`poolname`*`]`	Reserving connections in a connection pool.
`reset.weblogic.jdbc.connectionPool [.`*`poolname`*`]`	Resetting all the connections in a connection pool by shutting down and reestablishing all allocated connections.
`shrink.weblogic.jdbc.connectionPool [.`*`poolname`*`]`	Shrinking the connection pool to its original size (number of connections).

Table 8-1 File Realm JDBC ACLs

Use this ACL . . .	To Restrict . . .
`admin.weblogic.jdbc.connectionPool [.poolname]`	Enabling, disabling, and shutting down the connection pool.
`admin.weblogic.jdbc.connectionPoolcreate`	Creation of static connection pools.

Configuring and Managing JDBC Connection Pools, MultiPools, and DataSources Using the Administration Console

The following sections discuss how to set database connectivity by configuring JDBC components—connection pools, Data Sources, and MultiPools. Once connectivity is established, you use either the Administration Console or command-line interface to manage and monitor connectivity. See Table 8-3 for descriptions of the configuration tasks and links to the Administration Console Online Help.

JDBC settings you make in the Administration Console, including configuration settings for connection pools, MultiPools, DataSources, and TxDataSources, are persisted in the `config.xml` file for the domain. For information about entries in this file, see the following sections of the *Configuration Reference Guide*:

- JDBCConnectionPool at `http://e-docs.bea.com/wls/docs70/config_xml/JDBCConnectionPool.html`
- JDBCMultiPool at `http://e-docs.bea.com/wls/docs70/config_xml/JDBCMultiPool.html`
- JDBCDataSource at `http://e-docs.bea.com/wls/docs70/config_xml/JDBCDataSource.html`

- JDBCTxDataSource at `http://e-docs.bea.com/wls/docs70/config_xml/JDBCTxDataSource.html`

JDBC Configuration

In this section, we define *configuration* as including these processes:

Creating the JDBC Objects

Using the Administration Console, you create the JDBC components—connection pools, Data Sources, and MultiPools—by specifying attributes and database properties. See "Configuring JDBC Connectivity Using the Administration Console" on page 8-13.

First you create the connection pools and optionally a MultiPool, then you create the Data Source. When you create a Data Source object, you specify a connection pool or MultiPool as one of the Data Source attributes. This associates that Data Source with one specific connection pool or MultiPool ("pool").

Targeting the JDBC Objects

Once you configure the Data Source and associated connection pool (or MultiPool), you then target each object to the same servers or clusters. Some common scenarios are as follows:

- In a single server configuration, assign each Data Source and its associated connection pool to the server.
- In a cluster, assign the Data Source and related connection pool to the cluster, not to individual servers in the cluster.
- If you are using a MultiPool, assign the connection pools to the MultiPool; then assign the Data Source and all connection pools and the MultiPool to the same targets: server(s) or cluster(s).

See "JDBC Connections" in *Using WebLogic Clusters* at `http://e-docs.bea.com/wls/docs70/cluster/overview.html#jdbc_connections` for more information about connection pools, data sources, and MultiPools in a cluster. See "Configuring JDBC Connectivity Using the Administration Console" on page 8-13 for a description of the tasks you perform.

Refer to the following table for more information about association and assignment in the configuration process.

Table 8-2 Association and Assignment Scenarios

Scenario #	Associated . . .	Assign . . .	Target Description
1	Data Source A with Connection Pool A	1. Data Source A to Managed Server 1, and 2. Connection Pool A to Managed Server 1.	Data Source and connection pool assigned to the same target.
2	Data Source B with Connection Pool B	1. Data Source B to Cluster X, then 2. Connection Pool B to Managed Server 2 in Cluster X.	Data Source and Connection assigned to related server/cluster targets.
3	Data Source C with Connection Pool C	■ Data Source C and Connection Pool C to Managed Server 1. - AND - ■ Data Source C to Cluster X; then assign Connection Pool C to Managed Server 2 in Cluster X.	Data Source and connection pool assigned as a unit to two different targets.

You can assign these Data Source/connection pool combinations to more than one server or cluster, but they must be assigned in combination. For example, you cannot assign a Data Source to Managed Server A if its associated connection pool is assigned only to Server B.

You can configure dynamic connection pools (after the server starts) using the WebLogic API (see Creating a Dynamic Connection Pool in *Programming WebLogic JDBC*) or the command-line interface (see "JDBC Configuration Tasks Using the Command-Line Interface" on page 8-16). WebLogic Server also includes example code for creating and configuring dynamic Data Sources and connection pools in the server samples, if you opted to install samples during installation. See *`SAMPLES_HOME`*`\server\src\examples\jdbc`, where *`SAMPLES_HOME`* is the location of the top-level directory for all samples and examples for the WebLogic Platform (`c:\bea\weblogic700\samples`, by default).

Configuring JDBC Connectivity Using the Administration Console

The Administration Console allows you to configure, manage, and monitor JDBC connectivity. To display the tabs that you use to perform these tasks, follow these steps:

1. Start the Administration Console.
2. Locate the Services node in the left pane, then expand the JDBC node.
3. Select the node in the tree specific to the component you want to configure or manage—connection pools, MultiPools, Data Source, or Tx Data Source.
4. Follow the instructions in the Online Help. For links to the Online Help, see Table 8-3.

The following table shows the connectivity tasks, listed in typical order in which you perform them. You may perform these tasks in a different order; but you must configure an object before associating or assigning it.

Table 8-3 JDBC Configuration Tasks

JDBC Component/ Task	Description
Configure a Connection Pool	On the Configuration tabs in the right pane, you set the attributes for the connection pool, such as Name, URL, and database Properties.

Table 8-3 JDBC Configuration Tasks

JDBC Component/ Task	Description
Clone a Connection Pool (Optional)	This task copies a connection pool. On the Configuration tabs, you change Name of pool to a unique name; and accept or change the remaining attributes. This a useful feature when you want to have identical pool configurations with different names. For example, you may want to have each database administrator use a certain pool to track individual changes to a database.
Assign a Connection Pool to the Servers/Clusters	Using the Target tab, you assign the connection pool to one or more Servers or Clusters. See Table 8-2 Association and Assignment Scenarios. Also, to assign several connection pools to a server, see Assigning JDBC Connection Pools to a Server in the Online Help.
Configure a MultiPool (Optional)	On the MultiPool tabs, you set the attributes for the name and algorithm type, either High Availability or Load Balancing. On the Pool tab, you assign the connection pools to this MultiPool.
Assign the MultiPool to Servers or Clusters	Using the Target tab, you assign the configured MultiPool to Servers or Clusters.
Configure a Data Source (and Associate it with a Connection Pool)	Using the Data Source tab, set the attributes for the Data Source, including the Name, JNDI Name, and Pool Name (this associates, or assigns, the Data Source with a specific pool—connection pool or MultiPool.)
Assign the Data Source to Servers or Clusters	Using the Target tab, you assign the configured Data Source to servers or clusters.
Configure a Tx Data Source (and Associate it with a Connection Pool)	Using the Tx Data Source tab, set the attributes for the Tx Data Source, including the Name, JNDI Name, and *Connection Pool* Name (this associates, or assigns, the Data Source with a specific pool). **Note:** Do not associate a Tx Data Source with a MultiPool; MultiPools are not supported in distributed transactions.
Assign the Tx Data Source to Servers	Using the Target tab, you assign the configured Tx Data Source to servers.

Database Passwords in Connection Pool Configuration

When you create a connection pool, you typically include at least one password to connect to the database. If you use an open string to enable XA, you may use two passwords. You can enter the passwords as a name-value pair in the `Properties` field or you can enter them in their respective fields:

- **`Password`**. Use this field to set the database password. This value overrides any `password` value defined in the `Properties` passed to the tier-2 JDBC Driver when creating physical database connections. The value is encrypted in the `config.xml` file (stored as the `Password` attribute in the `JDBCConnectionPool` tag) and is hidden on the administration console.
- **`Open String Password`**. Use this field to set the password in the open string that the transaction manager in WebLogic Server uses to open a database connection. This value overrides any password defined as part of the open string in the `Properties` field. The value is encrypted in the `config.xml` file (stored as the `XAPassword` attribute in the `JDBCConnectionPool` tag) and is hidden on the Administration Console. At runtime, WebLogic Server reconstructs the open string with the password you specify in this field. The open string in the Properties field should follow this format:

  ```
  openString=Oracle_XA+Acc=P/userName/+SesTm=177+DB=dbHost+Thread
  s=true=Sqlnet=dvi0+logDir=.
  ```

  Note that after the `userName` there is no password.

If you specify a password in the `Properties` field when you first configure the connection pool, WebLogic Server removes the password from the `Properties` string and sets the value as the `Password` value in an encrypted form the next time you start WebLogic Server. If there is already a value for the `Password` attribute for the connection pool, WebLogic Server does not change any values. However, the value for the `Password` attribute overrides the password value in the `Properties` string. The same behavior applies to any password that you define as part of an open string. For example, if you include the following properties when you first configure a connection pool:

```
user=scott;
password=tiger;
openString=Oracle_XA+Acc=p/scott/tiger+SesTm=177+db=dbHost+Thread
s=true+Sqlnet=lcs817+logDir=.+dbgFl=0x15;server=dbHost
```

The next time you start WebLogic Server, it moves the database password and the password included in the open string to the Password and Open String Password attributes, respectively, and the following value remains for the `Properties` field:

```
user=scott;
openString=Oracle_XA+Acc=p/scott/+SesTm=177+db=dbHost+Threads=tru
e+Sqlnet=lcs817+logDir=.+dbgFl=0x15;server=dbHost
```

After a value is established for the `Password` or `Open String Password` attributes, the values in these attributes override the respective values in the `Properties` attribute. That is, continuing with the previous example, if you specify `tiger2` as the database password in the `Properties` attribute, WebLogic Server ignores the value and continues to use `tiger` as the database password, which is the current encrypted value of the `Password` attribute. To change the database password, you must change the `Password` attribute.

Note: The value for `Password` and `Open String Password` do not need to be the same.

JDBC Configuration Tasks Using the Command-Line Interface

The following table shows what methods you use to create a dynamic connection pool.

Table 8-4 Setting Connectivity for Dynamic JDBC Connection Pools

If you want to . . .	Then use the . . .
Create a dynamic connection pool	■ Command line—"CREATE_POOL" on page B-41, or ■ API—see Configuring and Administering WebLogic JDBC in Programming WebLogic JDBC

For more information, see "WebLogic Server Command-Line Interface Reference" on page B-1, and "Creating a Connection Pool Dynamically" in Programming WebLogic JDBC.

Managing and Monitoring Connectivity

Managing connectivity includes enabling, disabling, and deleting the JDBC components once they have been established.

JDBC Management Using the Administration Console

To manage and monitor JDBC connectivity, refer to the following table:

Table 8-5 JDBC Management Tasks

If you want to . . .	Do this . . . in the Administration Console
Reassign a Connection Pool to a Different Server or Cluster	Using the instructions in Assign a Connection Pool to the Servers/Clusters, on the Target tab deselect the target (move target from Chosen to Available) and assign a new target. To assign several connection pools to a server, see Assigning JDBC Connection Pools to a Server in the Online Help.
Reassign a MultiPool to a Different Cluster	Using the instructions in Assign the MultiPool to Servers or Clusters, on the Target tab deselect the target (move target from Chosen to Available) and assign a new target.
Delete a Connection Pool	See Deleting a Connection Pool in the Online Help.
Delete a MultiPool	See Deleting a JDBC MultiPool in the Online Help.
Delete a Data Source	See Deleting a Connection Pool in the Online Help.
Monitor a Connection Pool	To monitor the connections for a single connection pool, see Monitoring Connections in a JDBC Connection Pool in the Online Help. To monitor all active connection pools for a server, see Monitoring All Active JDBC Connection Pools in the Online Help.

Table 8-5 JDBC Management Tasks

If you want to . . .	Do this . . . in the Administration Console
Modify an Attribute for a Connection Pool, MultiPool, or DataSource	1. Select the JDBC object—connection pool, MultiPool, or DataSource—in the left pane. 2. Select the Target tab in the right pane, and unassign the object from each server and cluster (move the object from the Chosen column to the Available column.) Then click Apply. This stops the JDBC object—connection pool, MultiPool, or DataSource—on the corresponding server(s). 3. Select the tab you want to modify and change the attribute. 4. Select the Target tab and reassign the object to the server(s).This starts the JDBC object—connection pool, MultiPool, or DataSource—on the corresponding server(s).

JDBC Management Using the Command-Line Interface

The following table describes the connection pool management using the command-line interface. Select the command for more information.

For information on using the connection pool commands, see "WebLogic Server Command-Line Interface Reference" on page B-1

Table 8-6 Managing Connection Pools with the Command Line Interface

If you want to . . .	Then use this command . . .
Disable a Connection Pool	DISABLE_POOL
Enable a Connection Pool that has been disabled	ENABLE_POOL
Delete a Connection Pool	DESTROY_POOL
Confirm if a Connection Pool was created	EXISTS_POOL
Reset a Connection Pool	RESET_POOL

JDBC Configuration Guidelines for Connection Pools, MultiPools, and DataSources

This section describes JDBC configuration guidelines for connection pools, MultiPools, and Data Sources used in local and distributed transactions.

Overview of JDBC Configuration

To set up JDBC connectivity, you configure connection pools, Data Source objects (always recommended, but optional in some cases), and MultiPools (optional) by defining attributes in the Administration Console or, for dynamic connection pools, in application code or at the command line.

There are three types of transaction scenarios:

- Local transactions—non-distributed transactions
- Distributed transactions using an XA Driver—distributed transactions with multiple participants that use two-phase commit
- Distributed transactions using a non-XA Driver—transactions with a single resource manager and single database instance that emulate two-phase commit

You configure Data Source objects (DataSources and TxDataSources), connection pools, and MultiPools according to the way transactions are handled in your system. The following table summarizes how to configure these objects for use in the three transaction scenarios:

Table 8-7 Summary of JDBC Configuration Guidelines

Description/ Object	**Local Transactions**	**Distributed Transactions XA Driver**	**Distributed Transactions Non-XA Driver**
JDBC driver	■ WebLogic jDriver for Oracle and Microsoft SQL Server. ■ Compliant third-party drivers.	■ WebLogic jDriver for Oracle/XA. ■ Compliant third-party drivers.	■ WebLogic jDriver for Oracle and Microsoft SQL Server ■ Compliant third-party drivers.
Data Source	Data Source object recommended. (If there is no Data Source, use the JDBC API.)	Requires Tx Data Source.	Requires Tx Data Source. Select Emulate Two-Phase Commit for non-XA Driver (set `enable two-phase commit=true)` if more than one resource is involved. See "Configuring Non-XA JDBC Drivers for Distributed Transactions" on page 8-35.
Connection Pool	Requires Data Source object when configuring in the Administration Console.	Requires Tx Data Source.	Requires Tx Data Source.
MultiPool	Connection Pool and Data Source required.	Not supported in distributed transactions.	Not supported in distributed transactions.

Note: For distributed transactions, use an XA-compliant driver, such as the *WebLogic jDriver for Oracle/XA*, which is the XA compliant version of the WebLogic jDriver for Oracle.

When to Use a Tx Data Source

If your applications or environment meet any of the following criteria, you should use a Tx Data Source instead of a Data Source:

- Use the Java Transaction API (JTA)
- Use the EJB container in WebLogic Server to manage transactions
- Include multiple database updates within a single transaction
- Access multiple resources, such as a database and the Java Messaging Service (JMS), during a transaction
- Use the same connection pool on multiple servers

With an EJB architecture, it is common for multiple EJBs that are doing database work to be invoked as part of a single transaction. Without XA, the only way for this to work is if all transaction participants use the exact same database connection. WebLogic Server uses the JTS driver and a TxDataSource (with Emulate Two-Phase Commit for non-XA Driver selected) to do this behind the scenes without requiring you to explicitly pass the JDBC connection from EJB to EJB. With XA (requires an XA driver), you can use a Tx Data Source in WebLogic Server for distributed transactions with two-phase commit so that EJBs can use a different database connections for each part of the transaction. In either case (with or without XA), you should use a Tx Data Source.

Read more about Data Sources in *Programming WebLogic JDBC* at http://e-docs.bea.com/wls/docs70/jdbc/programming.html.

Note: Do not create two Tx Data Sources that point to the same connection pool. If a transaction uses two different Tx Data Sources which are both pointed to the same connection pool, you will get an XA_PROTO error when you try to access the second connection.

Drivers Supported for Local Transactions

JDBC 2.0 drivers that support the JDBC Core 2.0 API (`java.sql`), including the WebLogic jDrivers for Oracle and Microsoft SQL Server. The API allows you to create the class objects necessary to establish a connection with a data source, send queries and update statements to the data source, and process the results.

Drivers Supported for Distributed Transactions Using XA

Any JDBC driver that supports JDBC 2.0 distributed transactions standard extension interfaces (`javax.sql.XADataSource`, `javax.sql.XAConnection`, `javax.transaction.xa.XAResource`), such as the WebLogic jDriver for Oracle/XA.

Drivers Supported for Distributed Transactions without XA

Any JDBC driver that supports JDBC 2.0 Core API but does not support JDBC 2.0 distributed transactions standard extension interfaces (non-XA). Only one non-XA JDBC driver can participate in a distributed transaction. See "Configuring Non-XA JDBC Drivers for Distributed Transactions" on page 8-35.

Avoiding Server Lockup with the Correct Number of Connections

When your applications attempt to get a connection from a connection pool in which there are no available connections, the connection pool throws an exception stating that a connection is not available in the connection pool. Connection pools do not queue requests for a connection. To avoid this error, make sure your connection pool can expand to the size required to accommodate your peak load of connection requests.

To set the maximum number of connections for a connection pool in the Administration Console, expand the navigation tree in the left pane to show the Services→JDBC→Connection Pools nodes and select a connection pool. Then, in the right pane, select the Configuration→Connections tab and specify a value for `Maximum Capacity`.

Configuring JDBC Drivers for Local Transactions

To configure JDBC drivers for local transactions, set up the JDBC connection pool as follows:

- Specify the Driver Classname attribute as the name of the class supporting the `java.sql.Driver` interface.
- Specify the data properties. These properties are passed to the specific `Driver` as driver properties. (Enter each on a separate line.)

For more information on WebLogic two-tier JDBC drivers, refer to the BEA documentation for the specific driver you are using: *Using WebLogic jDriver for Oracle* at `http://e-docs.bea.com/wls/docs70/oracle/index.html` and *Using WebLogic jDriver for Microsoft SQL Server* at `http://e-docs.bea.com/wls/docs70/mssqlserver4/index.html`. If you are using a third-party driver, refer to Using Third-Party JDBC XA Drivers with WebLogic Server in *Programming WebLogic JTA* at `http://e-docs.bea.com/wls/docs70/jta/thirdpartytx.html` and the vendor-specific documentation. The following tables show sample JDBC connection pool and Data Source configurations using the WebLogic jDrivers.

The following table shows a sample connection pool configuration using the WebLogic jDriver for Oracle.

Note: The following configuration examples use a Password attribute. The Password attribute value overrides any password defined in Properties (as a name/value pair). This attribute is passed to the 2-tier JDBC driver when creating physical database connections. The value is stored in an encrypted form in the `config.xml` file and can be used to avoid storing passwords in clear text in that file.

Table 8-8 WebLogic jDriver for Oracle: Connection Pool Configuration

Attribute Name	Attribute Value
General Tab	
Name	`myConnectionPool`
URL	`jdbc:weblogic:oracle`
Driver Classname	`weblogic.jdbc.oci.Driver`

Table 8-8 WebLogic jDriver for Oracle: Connection Pool Configuration

Attribute Name	Attribute Value

Properties	`user=scott`
	`server=localdb`

Password	`tiger` (Displayed as ***** when typed, hidden thereafter; this value overrides any password defined in Properties as a name value pair)
Connections Tab	
Initial Capacity	`1`
Max Capacity	`5`
Capacity Increment	`1`
Shrink Period	15
Testing Tab	
Test Table Name	dual
Targets Tab	
Targets	`myserver`

The following table shows a sample Data Source configuration using the WebLogic jDriver for Oracle or Microsoft SQL Server.

Table 8-9 Data Source Configuration

Attribute Name	Attribute Value
Configuration Tab	
Name	`myDataSource`
JNDI Name	`myconnection`
Pool Name	`myConnectionPool`
Row Prefetch Size	48

Table 8-9 Data Source Configuration

Attribute Name	Attribute Value
Stream Chunk Size	256
Targets Tab	
Targets	`myserver`

The following table shows a sample connection pool configuration using the WebLogic jDriver for Microsoft SQL Server.

Table 8-10 WebLogic jDriver for Microsoft SQL Server: Connection Pool Configuration

Attribute Name	Attribute Value
General Tab	
Name	`myConnectionPool`
URL	jdbc:weblogic:mssqlserver4
Driver Classname	`weblogic.jdbc.mssqlserver4.Driver`

Properties	`user=sa`
	`db=pubs`
	`server=myHost:1433`
	`appname=MyApplication`
	`hostname=myhostName`

Password	`secret` (Displayed as ****** when typed, hidden thereafter; this value overrides any password defined in Properties as a name value pair)
Connections Tab	
Initial Capacity	`1`
Max Capacity	`5`
Capacity Increment	`1`
Shrink Period	15

Table 8-10 WebLogic jDriver for Microsoft SQL Server: Connection Pool Configuration

Attribute Name	Attribute Value
Testing Tab	
Test Table Name	member
Targets Tab	
Targets	`myserver`

The following table shows a sample connection pool configuration using the IBM Informix JDBC Driver.

Table 8-11 IBM Informix JDBC Driver: Connection Pool Configuration

Attribute Name	Attribute Value
General Tab	
Name	`myConnectionPool`
URL	jdbc:informix-sqli:ifxserver:1543
Driver Classname	`com.informix.jdbc.IfxDriver`

Properties	`informixserver=ifxserver`
	`user=informix`

Password	`informix` (Displayed as ****** when typed, hidden thereafter; this value overrides any password defined in Properties as a name value pair)
Connections Tab	
Initial Capacity	`3`
Max Capacity	`10`
Capacity Increment	`1`
Login Delay Seconds	1
Shrink Period	15

Table 8-11 IBM Informix JDBC Driver: Connection Pool Configuration

Attribute Name	Attribute Value
Targets Tab	
Targets	`myserver`

Configuring XA JDBC Drivers for Distributed Transactions

To allow XA JDBC drivers to participate in distributed transactions, configure the JDBC connection pool as follows:

- Specify the `Driver Classname` attribute as the name of the class supporting the `javax.sql.XADataSource` interface.
- Make sure that the database properties are specified. These properties are passed to the specified `XADataSource` as data source properties. For more information on data source properties for the WebLogic jDriver for Oracle, see "WebLogic jDriver for Oracle/XA Data Source Properties." For information about data source properties for third-party drivers, see the vendor documentation.
- See "Additional XA Connection Pool Properties" on page 8-34 for any additional connection pool properties that may be required to support XA for your DBMS.

The following table shows an example of a JDBC connection pool configuration using the WebLogic jDriver for Oracle in XA mode.

Table 8-12 WebLogic jDriver for Oracle/XA: Connection Pool Configuration

Attribute Name	Attribute Value
General Tab	
Name	`fundsXferAppPool`
URL	*(none required)*
Driver Classname	`weblogic.jdbc.oci.xa.XADataSource`

Table 8-12 WebLogic jDriver for Oracle/XA: Connection Pool Configuration

Attribute Name	Attribute Value

Properties	`user=scott`
	`server=localdb`

Password	`tiger` (Displayed as ***** when typed, hidden thereafter; this value overrides any password defined in Properties as a name value pair)
Connections Tab	
Initial Capacity	`1`
Max Capacity	`5`
Capacity Increment	`1`
Shrink Period	15
Testing Tab	
Test Table Name	dual
Targets Tab	
Targets	`myserver`

The following table shows an example of a Tx Data Source configuration using the WebLogic jDriver for Oracle in XA mode.

Table 8-13 WebLogic jDriver for Oracle/XA: Tx Data Source

Attribute Name	Attribute Value
Configuration Tab	
Name	`fundsXferDataSource`
JNDI Name	`myapp.fundsXfer`
Pool Name	`fundsXferAppPool`
Targets Tab	

Table 8-13 WebLogic jDriver for Oracle/XA: Tx Data Source

Attribute Name	Attribute Value
Targets	`myserver`

You can also configure the JDBC connection pool to use a third-party vendor's driver in XA mode. In such cases, the data source properties are set via reflection on the `XADataSource` instance using the JavaBeans design pattern. In other words, for property `abc`, the `XADataSource` instance must support get and set methods with the names `getAbc` and `setAbc`, respectively.

The following attributes are an example of a JDBC connection pool configuration using the Oracle Thin Driver.

Table 8-14 Oracle Thin Driver: Connection Pool Configuration

Attribute Name	Attribute Value
General Tab	
Name	`jtaXAPool`
URL	`jdbc:oracle:thin:@server:port:sid`
Driver Classname	`oracle.jdbc.xa.client.OracleXADataSource`
Properties	`user=scott`
Password	`tiger` (Displayed as ***** when typed, hidden thereafter; this value overrides any password defined in Properties as a name value pair)
Connections Tab	
Initial Capacity	`4`
Max Capacity	`20`
Capacity Increment	`2`
Shrink Period	15
Testing Tab	
Test Table Name	dual

Table 8-14 Oracle Thin Driver: Connection Pool Configuration

Attribute Name	Attribute Value
Targets Tab	
Targets	`myserver`

The following table shows an example of a Tx Data Source configuration using the Oracle Thin Driver.

Table 8-15 Oracle Thin Driver: Tx Data Source Configuration

Attribute Name	Attribute Value
Configuration Tab	
Name	`jtaXADS`
JNDI Name	`jtaXADS`
Pool Name	`jtaXAPool`
Targets Tab	
Targets	`myserver`

The following table shows an example of a JDBC connection pool configuration for distributed transactions using the Pointbase JDBC driver.

Note: PointBase Server is an all-Java DBMS product included in the WebLogic Server distribution solely in support of WebLogic Server evaluation, either in the form of custom trial applications or through packaged sample applications provided with WebLogic Server. Non-evaluation development and/or production use of the Pointbase Server requires a separate license be obtained by the end user directly from PointBase.

Table 8-16 Pointbase: Connection Pool Configuration

Attribute Name	Attribute Value
General Tab	

Table 8-16 Pointbase: Connection Pool Configuration

Attribute Name	Attribute Value
Name	`demoXAPool`
URL	`jdbc:pointbase:server://localhost/demo`
Driver Classname	`com.pointbase.xa.xaDataSource`
Properties	`user=public` `DatabaseName=jdbc:pointbase:server://localhost/demo`
Password	`public` (Displayed as ****** when typed, hidden thereafter; this value overrides any password defined in Properties as a name value pair)
Connections Tab	
Initial Capacity	`2`
Max Capacity	`10`
Capacity Increment	`2`
Supports Local Transaction	`true`
Shrink Period	15
Testing Tab	
Test Table Name	users
Targets Tab	
Targets	`myserver`

Configure the Tx Data Source for use with a Pointbase driver as follows.

Table 8-17 Pointbase: Tx Data Source Configuration

Attribute Name	Attribute Value
Configuration Tab	

Table 8-17 Pointbase: Tx Data Source Configuration

Attribute Name	Attribute Value
Name	`jtaXADS`
JNDI Name	`JTAXADS`
Pool Name	`demoXAPool`
Targets Tab	
Targets	`myserver`

WebLogic jDriver for Oracle/XA Data Source Properties

Table 8-18 lists the data source properties supported by the WebLogic jDriver for Oracle. The JDBC 2.0 column indicates whether a specific data source property is a JDBC 2.0 standard data source property (S) or a WebLogic Server extension to JDBC (E).

The Optional column indicates whether a particular data source property is optional or not. Properties marked with Y* are mapped to the corresponding fields of the Oracle `xa_open` string (value of the `openString` property) as listed in Table 8-18. If they are not specified, their default values are taken from the `openString` property. If they are specified, their values should match those specified in the `openString` property. If the properties do not match, a `SQLException` is thrown when you attempt to make an XA connection.

Mandatory properties marked with N* are also mapped to the corresponding fields of the Oracle `xa_open` string. Specify these properties when specifying the Oracle `xa_open` string. If they are not specified or if they are specified but do not match, an `SQLException` is thrown when you attempt to make an XA connection.

Property Names marked with ** are supported but not used by WebLogic Server.

Table 8-18 Data Source Properties for WebLogic jDriver for Oracle/XA

Property Name	Type	Description	JDBC 2.0 standard/ extension	Optional	Default Value
databaseName**	String	Name of a particular database on a server.	S	Y	None
dataSourceName	String	A data source name; used to name an underlying XADataSource.	S	Y	Connection Pool Name
description	String	Description of this data source.	S	Y	None
networkProtocol**	String	Network protocol used to communicate with the server.	S	Y	None
password	String	A database password.	S	N*	None
portNumber**	Int	Port number at which a server is listening for requests.	S	Y	None
roleName**	String	The initial SQL role name.	S	Y	None
serverName	String	Database server name.	S	Y*	None
user	String	User’s account name.	S	N*	None
openString	String	Oracle’s XA open string.	E	Y	None
oracleXATrace	String	Indicates whether XA tracing output is enabled. If enabled (true), a file with a name in the form of `xa_poolnamedate.trc` is placed in the directory in which the server is started.	E	Y	false

Table 8-19 lists the mapping between Oracle's `xa_open` string fields and data source properties.

Table 8-19 Mapping of xa_open String Names to JDBC Data Source Properties

Oracle xa_open String Field Name	JDBC 2.0 Data Source Property	Optional
acc	user, password	N
sqlnet	ServerName	

Note: You must specify `Threads=true` in Oracle's `xa_open` string.

For a complete description of Oracle's `xa_open` string fields, see your Oracle documentation.

Additional XA Connection Pool Properties

When using connections from a connection pool in distributed transactions, you may need to set additional properties for the connection pool so that the connection pool handles the connection properly within WebLogic Server in the context of the transaction. You set these properties in the configuration file (`config.xml`) within the `JDBCConnectionPool` tag. By default, all additional properties are set to false. You set the properties to true to enable them.

In many cases, WebLogic Server automatically sets the proper value for these properties internally so that you do not have to set them manually.

KeepXAConnTillTxComplete

Some DBMSs require that you start and end a transaction in the same physical database connection. In some cases, a transaction in WebLogic Server may start in one physical database connection and end in another physical database connection. To force a connection pool to reserve a physical connection and provide the *same* connection to an application throughout transaction processing until the transaction is complete, you set `KeepXAConnTillTxComplete="true"`. For example:

```
<JDBCConnectionPool KeepXAConnTillTxComplete="true"
DriverName="com.sybase.jdbc2.jdbc.SybXADataSource"
CapacityIncrement="5" InitialCapacity="10" MaxCapacity="25"
Name="demoXAPool" Password="{3DES}vIF8diu4H0QmdfOipd4dWA=="
```

```
Properties="User=dbuser;DatabaseName=dbname;ServerName=server_nam
e_or_IP_address;PortNumber=serverPortNumber;NetworkProtocol=Tds;r
esourceManagerName=Lrm_name_in_xa_config;resourceManagerType=2" />
```

Note: This property is *required* to support distributed transactions with DB2 and Sybase.

Configuring Non-XA JDBC Drivers for Distributed Transactions

When configuring the JDBC connection pool to allow non-XA JDBC drivers to participate with other resources in distributed transactions, select the Emulate Two-Phase Commit for non-XA Driver attribute (`EnableTwoPhaseCommit` in the `JDBCTxDataSource` MBean) for the JDBC Tx Data Source. This parameter is ignored by resources that support the `XAResource` interface. Note that only one non-XA connection pool may participate in a distributed transaction.

Non-XA Driver/Single Resource

If you are using only one non-XA driver and it is the only resource in the transaction, leave the Emulate Two-Phase Commit for non-XA Driver option unselected in the Console (accept the default `EnableTwoPhaseCommit = false`). In this case, WebLogic Server ignores the setting and the Transaction Manager performs a one-phase optimization.

Non-XA Driver/Multiple Resources

If you are using one non-XA JDBC driver with other XA resources, select Emulate Two-Phase Commit in the Administration Console (`EnableTwoPhaseCommit = true`).

When the Emulate Two-Phase Commit for non-XA Driver option is selected (`EnableTwoPhaseCommit` is set to `true)`, the non-XA JDBC resource always returns `XA_OK` during the `XAResource.prepare()` method call. The resource attempts to commit or roll back its local transaction in response to subsequent `XAResource.commit()` or `XAResource.rollback()` calls. If the resource commit or rollback fails, a heuristic error results. Application data may be left in an inconsistent state as a result of a heuristic failure.

When the Emulate Two-Phase Commit for non-XA Driver option is not selected in the Console (`EnableTwoPhaseCommit` is set to `false`), the non-XA JDBC resource causes `XAResource.prepare()` to fail. This mechanism ensures that there is only one participant in the transaction, as `commit()` throws a `SystemException` in this case. When there is only one resource participating in a transaction, the one phase optimization bypasses `XAResource.prepare()`, and the transaction commits successfully in most instances.

This non-XA JDBC driver support is often referred to as the "JTS driver" because WebLogic Server uses the WebLogic JTS Driver internally to support the feature. For more information about the WebLogic JTS Driver, see "Using the WebLogic JTS Driver" in *Programming WebLogic JDBC*.

Limitations and Risks When Using a Non-XA Driver in Global Transactions

WebLogic Server supports the participation of non-XA JDBC resources in global transactions, but there are limitations that you must consider when designing applications to use such resources. Because a non-XA driver does not adhere to the XA/2PC contracts and only supports one-phase commit and rollback operations, WebLogic Server (through the JTS driver) has to make compromises to allow the resource to participate in a transaction controlled by the Transaction Manager.

Heuristic Completions and Data Inconsistency

When Emulate Two-Phase Commit is selected for a non-XA resource, (`enableTwoPhaseCommit = true`), the prepare phase of the transaction for the non-XA resource always succeeds. Therefore, the non-XA resource does not truly participate in the two-phase commit (2PC) protocol and is susceptible to failures. If a failure occurs in the non-XA resource after the prepare phase, the non-XA resource is likely to roll back the transaction while XA transaction participants will commit the transaction, resulting in a heuristic completion and data inconsistencies.

Because of the data integrity risks, the Emulate Two-Phase Commit option should only be used in applications that can tolerate heuristic conditions.

Cannot Recover Pending Transactions

Because a non-XA driver manipulates local database transactions only, there is no concept of a transaction pending state in the database with regard to an external transaction manager. When `XAResource.recover()` is called on the non-XA resource, it always returns an empty set of Xids (transaction IDs), even though there may be transactions that need to be committed or rolled back. Therefore, applications that use a non-XA resource in a global transaction cannot recover from a system failure and maintain data integrity.

Possible Performance Loss with Non-XA Resources in Multi-Server Configurations

Because WebLogic Server relies on the database local transaction associated with a particular JDBC connection to support non-XA resource participation in a global transaction, when the same JDBC data source is accessed by an application with a global transaction context on multiple WebLogic Server instances, the JTS driver will always route JDBC operations to the first connection established by the application in the transaction. For example, if an application starts a transaction on one server, accesses a non-XA JDBC resource, then makes a remote method invocation (RMI) call to another server and accesses a data source that uses the same underlying JDBC driver, the JTS driver recognizes that the resource has a connection associated with the transaction on another server and sets up an RMI redirection to the actual connection on the first server. All operations on the connection are made on the one connection that was established on the first server. This behavior can result in a performance loss due to the overhead associated with setting up these remote connections and making the RMI calls to the one physical connection.

Only One Non-XA Participant

When a non-XA resource (with Emulate Two-Phase Commit selected) is registered with the WebLogic Server Transaction Manager, it is registered with the name of the class that implements the XAResource interface. Since all non-XA resources with Emulate Two-Phase Commit selected use the JTS driver for the XAResource interface, all non-XA resources (with Emulate Two-Phase Commit selected) that participate in a global transaction are registered with the same name. If you use more than one non-XA resource in a global transaction, you will see naming conflicts or possible heuristic failures.

Non-XA Connection Pool and Tx Data Source Configuration Example

The following table shows configuration attributes for a sample JDBC connection pool using a non-XA JDBC driver.

Table 8-20 WebLogic jDriver for Oracle: Connection Pool Configuration

Attribute Name	Attribute Value
General Tab	
Name	`fundsXferAppPool`
URL	`jdbc:weblogic:oracle`
Driver Classname	`weblogic.jdbc.oci.Driver`

Properties	`user=scott`
	`server=localdb`

Password	`tiger` (Displayed as ***** when typed, hidden thereafter; this value overrides any password defined in Properties as a name value pair)
Connections Tab	
Initial Capacity	`0`
Max Capacity	`5`
Capacity Increment	`1`
Shrink Period	15
Testing Tab	
Test Table Name	dual
Targets Tab	
Targets	`myserver`

The following table shows configuration attributes for a sample Tx Data Source using a non-XA JDBC driver.

Table 8-21 WebLogic j Driver for Oracle: Tx Data Source Configuration

Attribute Name	Attribute Value
Configuration Tab	
Name	`fundsXferDataSource`
JNDI Name	`myapp.fundsXfer`
Pool Name	`fundsXferAppPool`
Emulate Two-Phase Commit for non-XA Driver	`selected (EnableTwoPhaseCommit = true)`
Targets Tab	
Targets	`myserver`

Increasing Performance with the Prepared Statement Cache

For each connection pool that you create in WebLogic Server, you can enable prepared statement caching by setting the prepared statement cache size or XA prepared statement cache size. When you enable prepared statement caching, WebLogic Server caches a set number of prepared and callable statements used in applications and EJBs. When an application or EJB calls any of the prepared or callable statements stored in the cache, WebLogic Server reuses the statement stored in the cache. Reusing statements eliminates the need for parsing statements in the database, which reduces CPU usage on the database machine, improving performance for the current statement and leaving CPU cycles for other tasks.

Statements are cached *per connection*, not per connection pool. For example, if you set the prepared statement cache size to 10, WebLogic Server will store 10 prepared statements called by applications or EJBs *that use that particular connection*.

With the release of WebLogic Server 7.0 Service Pack 3, the prepared statement cache feature was changed so that the cache behaves differently for connection pools that use an XA (transaction aware) JDBC driver to create database connections instead of a non-XA JDBC driver. You must set the appropriate cache size attribute depending on the type of JDBC driver used to create database connections in the JDBC connection pool: PreparedStatementCacheSize for connection pools that use a non-XA JDBC driver and XAPreparedStatementCacheSize for connection pools that use an XA JDBC driver. See "Non-XA Prepared Statement Cache" below and "XA Prepared Statement Cache" on page 8-41 for more details.

Non-XA Prepared Statement Cache

For the non-XA prepared statement cache, WebLogic Server uses a Fixed algorithm to determine which statements to store in the cache for each connection in the connection pool: WebLogic Server caches prepared and callable statements used on the connection until the prepared statement cache size is reached. When additional statements are used, they are not cached.

Note: You can use the JMX API to clear the statement cache. See the `clearStatementCache()` method in the Javadocs for WebLogic Classes at `http://e-docs.bea.com/wls/docs70/javadocs/weblogic/jdbc/extensions/WLConnection.html`.

This statement cache is used only for connection pools that use a non-XA JDBC driver to create database connections. If the connection pool uses an XA JDBC driver for database connections, the cache settings are ignored.

The default value for the non-XA prepared statement cache size is `5`. You can use the following methods to set the prepared statement cache size for a connection pool:

- Use the Administration Console to set the Prepared Statement Cache Size attribute for a connection pool. See Creating and Configuring a JDBC Connection Pool in the *Administration Console Online Help* at `http://e-docs.bea.com/wls/docs70/ConsoleHelp/jdbc.html#jdbc_conn_pool_create`.
- Use the WebLogic management API to set the PreparedStatementCacheSize attribute. See the `getPreparedStatementCacheSize()` and `setPreparedStatementCacheSize(int cacheSize)` methods in the Javadocs for WebLogic Classes at

`http://e-docs.bea.com/wls/docs70/javadocs/weblogic/management/configuration/JDBCConnectionPoolMBean.html`.

- Set the attribute directly in the configuration file (when Weblogic Server is not running).

To set the non-XA prepared statement cache size for a connection pool using the configuration file, before starting the server, open the `config.xml` file in an editor, then add an entry for the `PreparedStatementCacheSize` attribute in the `JDBCConnectionPool` tag. For example:

```
<JDBCConnectionPool CapacityIncrement="5"
    DriverName="com.pointbase.jdbc.jdbcUniversalDriver"
    InitialCapacity="5" MaxCapacity="20" Name="demoPool"
    Password="{3DES}ANfMduXgaaGMeS8+CR1xoA=="
    PreparedStatementCacheSize="20" Properties="user=examples"
    RefreshMinutes="0" ShrinkPeriodMinutes="15"
    ShrinkingEnabled="true" Targets="examplesServer"
    TestConnectionsOnRelease="false"
    TestConnectionsOnReserve="false"
    URL="jdbc:pointbase:server://localhost/demo"/>
```

XA Prepared Statement Cache

For the XA prepared statement cache, WebLogic Server uses a least recently used (LRU) algorithm to determine which statements to store in the cache for each connection in the connection pool: WebLogic Server caches prepared and callable statements used on the connection until the statement cache size is reached. When an application calls `Connection.prepareStatement()`, WebLogic Server checks to see if the statement is stored in the prepared statement cache. If so, WebLogic Server returns the cached statement (if it is not already being used). If the statement is not in the cache, and the cache is full (number of statements in the cache = XAPreparedStatementCacheSize), Weblogic Server determines which existing statement in the cache was the *least recently used* and replaces that statement in the cache with the new statement.

Note: You can use the JMX API to clear the statement cache. See the `clearStatementCache()` method in the Javadocs for WebLogic Classes at `http://e-docs.bea.com/wls/docs70/javadocs/weblogic/jdbc/extensions/WLConnection.html`.

The XA prepared statement cache is used only for connection pools that use an XA JDBC driver to create database connections. If the connection pool uses a non-XA JDBC driver for database connections, the cache settings are ignored.

The default value for the XA prepared statement cache size is `5`. You can use the following methods to set the XA prepared statement cache size for a connection pool:

- Use the Administration Console to set the XA Prepared Statement Cache Size attribute for a connection pool. See Creating and Configuring a JDBC Connection Pool in the *Administration Console Online Help* at `http://e-docs.bea.com/wls/docs70/ConsoleHelp/jdbc.html#jdbc_conn_pool_create`.
- Use the WebLogic management API to set the XAPreparedStatementCacheSize attribute. See the `getXAPreparedStatementCacheSize()` and `setXAPreparedStatementCacheSize(int cacheSize)` methods in the Javadocs for WebLogic Classes at `http://e-docs.bea.com/wls/docs70/javadocs/weblogic/management/configuration/JDBCConnectionPoolMBean.html`.
- Set the attribute directly in the configuration file (when Weblogic Server is not running).

To set the XA prepared statement cache size for a connection pool using the configuration file, before starting the server, open the `config.xml` file in an editor, then add an entry for the `PreparedStatementCacheSize` attribute in the `JDBCConnectionPool` tag. For example:

```
<JDBCConnectionPool CapacityIncrement="5"
    DriverName="com.pointbase.xa.xaDataSource"
    InitialCapacity="5" MaxCapacity="20" Name="demoXAPool"
    Password="{3DES}ANfMduXgaaGMeS8+CR1xoA=="
    XAPreparedStatementCacheSize="20"
    Properties="user=examples;
    DatabaseName=jdbc:pointbase:server://localhost/demo"
    RefreshMinutes="0" ShrinkPeriodMinutes="15"
    ShrinkingEnabled="true" Targets="examplesServer"
    TestConnectionsOnRelease="false"
    TestConnectionsOnReserve="false"
    URL="jdbc:pointbase:server://localhost/demo"/>
```

Usage Restrictions for the Prepared Statement Cache

Using the prepared statement cache can increase performance, but you must consider its limitations before you decide to use it. Please note the following restrictions when using the prepared statement cache. These restrictions apply to both the XA and non-XA prepared statement caches.

There may be other issues related to caching prepared statements that are not listed here. If you see errors in your system related to prepared statements, you should set the prepared statement cache size to `0`, which turns off prepared statement caching, to test if the problem is caused by caching prepared statements.

Calling a Stored Prepared Statement After a Database Change May Cause Errors

Prepared statements stored in the cache refer to specific database objects at the time the prepared statement is cached. If you perform any DDL (data definition language) operations on database objects referenced in prepared statements stored in the cache, the statements will fail the next time you run them. For example, if you cache a statement such as `select * from emp` and then drop and recreate the `emp` table, the next time you run the cached statement, the statement will fail because the exact `emp` table that existed when the statement was prepared, no longer exists.

Likewise, prepared statements are bound to the data type for each column in a table in the database at the time the prepared statement is cached. If you add, delete, or rearrange columns in a table, prepared statements stored in the cache are likely to fail when run again.

Using setNull In a Prepared Statement

When using the WebLogic jDriver for Oracle to connect to the database, if you cache a prepared statement that uses a `setNull` bind variable, you must set the variable to the proper data type. If you use a generic data type, as in the following example, the statement will fail when it runs with a value other than null.

```
java.sql.Types.Long sal=null

.
.
.
```

```
if (sal == null)
    setNull(2,int)//This is incorrect
else
    setLong(2,sal)
```

Instead, use the following:

```
if (sal == null)
    setNull(2,long)//This is correct
else
    setLong(2,sal)
```

This issue occurs consistently when using the WebLogic jDriver for Oracle. It may occur when using other JDBC drivers.

Prepared Statements in the Cache May Reserve Database Cursors

When WebLogic Server caches a prepared statement, the prepared statement may open a cursor in the database. If you cache too many statements, you may exceed the limit of open cursors for a connection. To avoid exceeding the limit of open cursors for a connection, you can change the limit in your database management system or you can reduce the prepared statement cache size for the connection pool.

Determining the Proper Prepared Statement Cache Size

To determine the optimum setting for the prepared statement cache size, you can emulate your server workload in your development environment and then run the Oracle statspack script. In the output from the script, look at the number of parses per second. As you increase the prepared statement cache size, the number of parses per second should decrease. Incrementally increase the prepared statement cache size until the number or parses per second no longer decreases.

Note: Consider the usage restrictions for the prepared statement cache before you decide to use it in your production environment. See "Usage Restrictions for the Prepared Statement Cache" on page 8-43 for more information.

Using a Startup Class to Load the Non-XA Prepared Statement Cache

To make the best use of the non-XA prepared statement cache and to get the best performance, you may want to create a startup class that calls each of the prepared statements that you want to store in the prepared statement cache. WebLogic Server caches prepared statements in the order that they are used and stops caching statements when it reaches the prepared statement cache size limit. By creating a startup class that calls the prepared statements that you want to cache, you can fill the cache with statements that your applications will reuse, rather than with statements that are called only a few times, thus getting the best performance increase with the least number of cached statements. You can also avoid caching prepared statements that my be problematic, such as those described in "Usage Restrictions for the Prepared Statement Cache" on page 8-43.

Even if the startup class fails, WebLogic Server loads and caches the statements for future use.

Note that each connection has it's own cache of statements. If you use a startup class to cache statements, you must create the class in such a way that it gets each connection from the pool and calls the prepared statements that you want to cache on each statement.

If you enable the connection pool to grow as demand for connections increases, new connections will cache statements as the statements are used. The startup class cannot load the prepared statement cache for new connections. If you enable the connection pool to shrink, the connection pool will close connections after the shrink period has been met and connections are available. There is now way to specify which connections to close first. Therefore, the connections for which you loaded the prepared statement cache may close before non-loaded connections close.

Also note that when starting a server, EJBs are deployed before startup classes are run. Prepared statements in CMP entity beans and prepared statements that EJBs use during deployment will be stored in the cache before prepared statements in a startup class. To work around this, you can clear the prepared statement cache for your connection pool after all EJBs and applications are deployed, and then run cache-priming code. See the Javadoc for the `clearStatementCache` method in the `weblogic.jdbc.extensions` package at `http://e-docs.bea.com/wls/docs70/javadocs/weblogic/jdbc/extensions/WLConnection.html`.

Because the XA prepared statement cache uses a Least Recently Used algorithm to replace statements in the cache, preloading the cache with a startup class will not be effective.

CHAPTER

9 Managing JMS

The following sections explain how to manage the Java Message Service (JMS) for WebLogic Server:

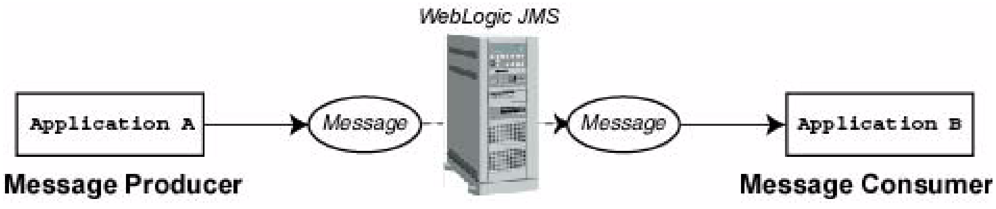
JMS and WebLogic Server

JMS is a standard API for accessing enterprise messaging systems. Specifically, WebLogic JMS:

- Enables Java applications sharing a messaging system to exchange messages.
- Simplifies application development by providing a standard interface for creating, sending, and receiving messages.

The following figure illustrates WebLogic JMS messaging.

Figure 9-1 WebLogic Server JMS Messaging



As illustrated in the figure, WebLogic JMS accepts messages from *producer* applications and delivers them to *consumer* applications.

Configuring JMS

Using the Administration Console, you define configuration attributes to:

- Create JMS servers and target a WebLogic Server instance or a Migratable Target where the JMS server will be deployed.
- Create and/or customize values for JMS servers, connection factories, destinations (physical queues and topics), distributed destinations (sets of physical queue and topic members within a cluster), destination templates, destination sort order (using destination keys), persistent stores, paging stores, session pools, and connection consumers.
- Set up custom JMS applications.
- Define thresholds and quotas.
- Enable any desired JMS features, such as server clustering, concurrent message processing, destination sort ordering, persistent messaging, paging, flow control, and load balancing for distributed destinations.

WebLogic JMS provides default values for some configuration attributes; you must provide values for all others. If you specify an invalid value for any configuration attribute, or if you fail to specify a value for an attribute for which a default does not exist, WebLogic Server will not boot JMS when you restart it. A sample `examplesJMSServer` configuration is provided with the product in the Examples

Server. For more information about starting the Examples Server, see "Starting the Default, Examples, and Pet Store Servers" in the *Installation Guide*. There are also instructions for manually configuring a basic JMS implementation in "Starting WebLogic Server and Configuring JMS" on page 9-5.

When you port WebLogic JMS applications from a previous release of Weblogic Server, the configuration information is automatically converted, as described in "Porting WebLogic JMS Applications" in *Programming WebLogic JMS*.

To configure WebLogic JMS attributes, follow the procedures described in the following sections, or in the *Administration Console Online Help*, to create and configure the JMS objects.

Once WebLogic JMS is configured, applications can send and receive messages using the JMS API. For more information about developing WebLogic JMS applications, refer to "Developing a WebLogic JMS Application" in *Programming WebLogic JMS*.

Note: To assist with your WebLogic JMS configuration planning, *Programming WebLogic JMS* provides configuration checklists for the attribute requirements and/or options that support various JMS features.

JMS Resource Naming Rules for Domain Interoperability

Within a WebLogic domain, each server instance, machine, cluster, virtual host, and any other resource type must be named uniquely and must not use the same name as the domain. These unique naming rules also apply to all JMS resources in either a single WebLogic domain or in a multi-domain environment.

Naming Rules for JMS Resources In a Single Domain Environment

For WebLogic JMS intra-domain interoperability, either between standalone or clustered server instances, the unique resource naming rule applies to all configurable JMS objects, such as JMS servers, stores (file or JDBC), templates, connection factories, session pools, and connection consumers. For example, you cannot have two similarly named JMS servers (e.g., *myJMSServer*) targeted to two different server instances within a domain.

For more information on how JMS clustering works, refer to "Configuring WebLogic JMS Clustering" in *Programming WebLogic JMS*

An exception to this unique naming rule within a domain, however, is for JMS queue and topic destinations on different JMS servers in a domain, as follows:

- Queue destinations *can* use the same name as other queues on different JMS servers; topic destinations can also use the same name as other topics on different JMS servers.
- Queue destinations *cannot* use the same name as topic destinations, nor can queues nor topics use the same name as any other configurable objects.

Naming Rules for JMS Resources In a Multi-Domain Environment

With the exception of JMS connection factories, the single domain naming rules for WebLogic JMS resources also apply to achieve successful inter-domain operability when using the Messaging Bridge to access other WebLogic domains. This rule includes domains from different releases of WebLogic Server.

For example, you cannot create a WebLogic Server instance named *myserver* in a version 7.0 domain named *mydomain70* if there is already a server instance named *myserver* in a version 6.1 domain named *mydomain61*. Similarly, on the JMS subsystem level, you cannot have two JMS servers or stores with the same name, even when they reside in separate domains.

Therefore, you must adhere to the following rules when interoperating between WebLogic domains:

- All domain names must be unique across your enterprise.
- All WebLogic Server instance names must be unique within a domain and across all interoperating domains.
- All JMS server names must be unique within a domain and across all interoperating domains.
- If a JMS store (file or JDBC) is being used for persistent messages, all store names must be unique within a domain and across all interoperating domains.

For detailed information about configuring a Messaging Bridge to interoperate with WebLogic domains, see "Using the Messaging Bridge to Interoperate with Different WebLogic Server Releases and Domains" on page 10-17.

Starting WebLogic Server and Configuring JMS

The following sections review how to start WebLogic Server and the Administration console, as well as provide a procedure for configuring a basic JMS implementation.

Starting the Default WebLogic Server

The default role for a WebLogic Server is the Administration Server. If a domain consists of only one WebLogic Server, that server is the Administration Server. If a domain consists of multiple WebLogic Servers, you must start the Administration Server first, and then you start the Managed Servers.

For complete information about starting the Administration Server, see "Starting an Administration Server" on page 2-12.

Starting the Administration Console

The Administration Console is the Web-based administrator front-end (administrator client interface) to WebLogic Server. You must start the server before you can access the Administration Console for a server.

For complete details about using the Administration Console to configure a WebLogic Server, see "Starting and Using the Administration Console" on page 1-22.

Configuring a Basic JMS Implementation

This section describes how to configure a basic JMS implementation using the Administration Console.

1. Under the Services node in the left pane, click the JMS node to expand the list.
2. For storing persistent messages in a disk-based file, you must create a JMS file store. If your are using the JMS Paging feature, you must also configure an additional "paging" JMS file store for temporarily swapping message bodies out to disk when memory is exhausted.
    a. Create a directory on the file system where the JMS file store will be kept.
    b. Click the Stores node in the left pane, and then click the Configure a new JMS File Store link in the right pane.

   c. On the General tab, give the store a name, specify a directory, and optionally, select a Synchronous Write policy to determine how the file store writes data to disk. Then click Create.

   d. Repeat these steps to create a Paging Store.

   For more information on configuring stores, see "Configuring Stores" on page 9-15.

3. For storing persistent messages in a JDBC-accessible database, you must create a JMS JDBC store. (If you need to create a JDBC connection pool, then first complete steps A–D; otherwise, you can skip to step E.) If your are using the JMS Paging feature, you must also configure a "paging" JMS file store for temporarily swapping message bodies out to disk.

   a. Click the JDBC node in the left pane to expand it.

   b. Click the Connection Pools node in the left pane, and then click the Configure a new JDBC Connection Pool link in the right pane.

   c. On the Configuration tabs, set the attributes for the connection pool, such as Name, URL, and database Properties. Click Apply on each tab when you're done making changes.

   d. On the Targets tab, target a WebLogic Server instance or a server cluster on which to deploy the connection pool by selecting either the Servers tab or the Clusters tab. Select a target by moving it from the Available list into the Chosen List, and then click Apply.

   e. Return to the JMS –> Stores node, and then click the Configure a new JMSJDBCStore link in the right pane.

   f. Give the JDBC Store a name, select a connection pool, and a prefix name. Then click Create.

   For more information on using JMS JDBC stores, see "About JMS JDBC Stores" on page 9-16. For more information on configuring JDBC connection pools, see "Configuring and Managing JDBC Connection Pools, MultiPools, and DataSources Using the Administration Console" on page 8-10.

4. Optionally, create a JMS Template to define multiple destinations with similar attribute settings. You also need a JMS Template to create temporary queues.

   a. Click the Templates node in the left pane, and then click the Configure a new JMS Template link in the right pane.

   b. On the General tab, give the template a name, and then click Create.

   c. Fill in the Thresholds & Quotas, Override, and Redelivery tabs, as appropriate. Click Apply on each tab when you're done making changes.

   For more information on configuring a JMS Template, see "Configuring JMS Templates" on page 9-13.

5. Configure a JMS Server, as follows:

   a. Click the Server node in the left pane, and then click the Configure a new JMSServer link in the right pane.

   b. On the General tab, give the server a name, select a Store if you created one, select a Paging Store if you created one, and select a Template if you created one. Then click Create.

   c. Fill in the Thresholds & Quotas tab, as appropriate. Click Apply when you're done making changes.

   d. On the Targets tab, target a WebLogic Server instance or a Migratable Target server on which to deploy the JMS server by selecting either the Servers tab or the Migratable Targets tab. Select a target by moving it from the Available list into the Chosen List, and then click Apply.

   For more information on configuring a JMS Server, see "Configuring JMS Servers" on page 9-8.

6. Create the JMS Destinations, which are queues (Point-To-Point) or topics (Pub/Sub):

   a. Under the Servers node in the left pane, click your new JMS server instance to expand the list, and then click the Destinations node.

   b. Click either the Configure a new JMSQueue or Configure a new JMSTopic link in the right pane.

   c. On the General tab, give the destination a name and a JNDI name. Fill in the other attributes, as appropriate, and then click Create.

   d. Fill in the Thresholds & Quotas, Override, Redelivery, and Multicast (topics only) tabs, as appropriate. Click Apply on each tab when you're done making changes.

   For more information on configuring a Destinations, see "Configuring Destinations" on page 9-12.

7. Create a Connection Factory to enable your JMS clients to create JMS connections:

   a. Click to the expand the Connection Factory node and in the left pane, and then click the Configure a new JMS Connection Factory link in the right pane.

   b. On the General tab, give the connection factory a name and a JNDI name. Fill in the other attributes, as appropriate, and then click Create.

   c. Fill in the Transactions and Flow Control tabs, as appropriate. Click Apply on each tab when you're done making changes.

   d. On the Targets tab, target a WebLogic Server instance or a server cluster on which to deploy the connection factory by selecting either the Servers tab or the Clusters tab. Select a target by moving it from the Available list into the Chosen List, and then click Apply.

   For more information on configuring a Connection Factory, see "Configuring Connection Factories" on page 9-10.

8. Optionally, use the Destination Keys node to define the sort order for a specific destination. For more information, see "Configuring Destination Keys" on page 9-14.

9. Optionally, use the Distributed Destinations node to make your physical destinations part of a logical distributed destination set within a server cluster. For more information, see "Configuring Distributed Destinations" on page 9-44.

10. Optionally, create JMS Session Pools, which enable your applications to process messages concurrently, and Connection Consumers (queues or topics) that retrieve server sessions and process messages. For more information, see "Configuring Session Pools" on page 9-19 and "Configuring Connection Consumers" on page 9-20.

Configuring JMS Servers

A JMS server manages connections and message requests on behalf of clients. To create a JMS server, use the Servers node in the Administration Console and define the following:

- General configuration attributes, including:

- Name of the JMS server.
- Persistent store (file or JDBC database) required for persistent messaging. If you do not assign a persistent store for a JMS server, persistent messaging is not supported on that server. For more information on configuring JMS stores, see "Configuring Stores" on page 9-15.
- Paging store (file store recommended) required for paging. If you do not assign a paging store for a JMS server, paging is not support on that server. For more information of configuring paging, see "Using Message Paging" on page 9-27.
- Temporary template that is used to create all temporary destinations, including temporary queues and temporary topics.

Note: JMS servers must be uniquely named within a domain. For more information, see "JMS Resource Naming Rules for Domain Interoperability" on page 9-3.

- Thresholds and quotas for messages and bytes (maximum number, and high and low thresholds), and whether or not bytes paging and/or messages paging is enabled.
- Target a WebLogic Server instance or a Migratable Target on which to deploy a JMS server.
  - Servers – When a target WebLogic Server boots, the JMS server boots as well. If no target WebLogic Server is specified, the JMS server will not boot.

Note: The deployment of a JMS server differs from that of a connection factory or template. A JMS server is deployed on a single WebLogic Server instance or on a migratable target (see next bullet item); whereas, a connection factory or template can be instantiated on multiple WebLogic Server instances simultaneously.

  - Migratable Targets – Migratable targets define a set of WebLogic Server instances in a cluster that can potentially host an "exactly-once" service, such as JMS. When a migratable target server boots, the JMS server boots as well on the *user-preferred* server in the cluster. However, a JMS server and all of its destinations can migrate to another server within the cluster in response to a WebLogic Server failure or due to a scheduled migration or system maintenance. For more information on configuring a migratable target for JMS, see "Managing JMS" in *Programming WebLogic JMS*.

For instructions on creating and configuring a JMS server, see "JMS Server Tasks" in the *Administration Console Online Help*.

Configuring Connection Factories

Connection factories are objects that enable JMS clients to create JMS connections. A connection factory supports concurrent use, enabling multiple threads to access the object simultaneously. WebLogic JMS provides a preconfigured default connection factory, `weblogic.jms.ConnectionFactory`, that can be disabled on a per-server basis, as described in "Server --> Services --> JMS" in the *Administration Console Online Help*. Otherwise, you can configure one or more connection factories to create connections with predefined attributes that better suit your application — as long as each connection factory is uniquely named. WebLogic Server adds them to the JNDI space during startup, and the application then retrieves a connection factory using WebLogic JNDI.

You can establish cluster-wide, transparent access to destinations from any server in a cluster either by enabling the default connection factory for each server instance, or by configuring one or more connection factories and targeting them to one or more server instances in the cluster. This way, each connection factory can be deployed on multiple WebLogic Servers. For more information on configuring JMS clustering, see "Managing JMS" in *Programming WebLogic JMS*.

To configure connection factories, use the Connection Factories node in the Administration Console to define the following:

- General configuration attributes, including:
  - Name of the connection factory.
  - Name for accessing the connection factory within the JNDI namespace.
  - Client identifier (client ID) for clients with durable subscribers. For more information about durable subscribers, see "Developing a WebLogic JMS Application" in *Programming WebLogic JMS*.
  - Default message delivery attributes (Priority, Time To Live, Time To Deliver, and Delivery Mode).
  - Maximum number of outstanding messages that may exist for an asynchronous session and the overrun policy (that is, the action to be taken, for multicast sessions, when this maximum is reached).

- Whether or not the `close()` method is allowed to be called from the `onMessage()` method.
- Whether all messages or only previously received messages are acknowledged.
- For distributed destinations, determine whether non-anonymous producers created through a connection factory are load balanced on a per-call basis.
- For distributed destinations, determine whether server affinity is used when load balancing consumers or producers in a distributed destination.

Note: JMS connection factories must be uniquely named within a domain. For more information, see "JMS Resource Naming Rules for Domain Interoperability" on page 9-3.

- Transaction attributes—Transaction time-out, whether Java Transaction API (JTA) user transactions are allowed, whether a transaction (XA) queue or XA topic connection factory is returned, and whether server-side transactions are enabled.
- Flow Control attributes—Allow a message producer to adjust its message flow. For more information about configuring flow control, see "Establishing Message Flow Control" on page 9-38.
- Target a WebLogic Server instance or a server cluster. Targets enable you to limit the set of servers, groups, and/or clusters on which a connection factory may be deployed.
  - Server—Target a single WebLogic Server instance on which to deploy a connection factory.
  - Cluster—Target a cluster on which to deploy a connection factory, in order to support cluster-wide, transparent access to destinations from any server in the cluster.

For the default connection factory, `weblogic.jms.ConnectionFactory`, all configuration attributes are set to their default values. If the default connection factory definition is appropriate for your application, you do not need to configure any additional connection factories for your application.

Note: Using the default connection factory, you have no control over the WebLogic Server on which the connection factory may be deployed. If you would like to target a particular server, create a new connection factory and specify the appropriate server target(s).

For instructions on creating and configuring a connection factory, see “Connection Factory Tasks” in the *Administration Console Online Help*.

Some connection factory attributes are dynamically configurable. When dynamic attributes are modified at run time, the new values become effective for new connections only, and do not affect the behavior of existing connections.

Configuring Destinations

A destination identifies a queue (Point-To-Point) or a topic (Pub/Sub) for a JMS server. After defining a JMS server, configure one or more destination for each JMS server.

You configure destinations explicitly or by configuring a destination template that can be used to define multiple destinations with similar attribute settings, as described in “Configuring JMS Templates” on page 9-13.

Note: You can also configure multiple physical destinations as members of a single distributed destination set within a cluster. Therefore, if one instance within the cluster fails, then other instances of the same destination will be able to provide service to JMS producers and consumers. For more information, see “Configuring Distributed Destinations” on page 9-44.

To configure destinations explicitly, use the Destinations node in the Administration Console to define the following configuration attributes:

- General configuration attributes, including:
  - Name and type (queue or topic) of the destination.
  - Name for accessing the destination within the JNDI namespace.
  - Whether or not a store is enabled for storing persistent messages.
  - The JMS template used for creating destinations.
  - Keys used to define the sort order for a specific destination.

  Note: JMS destinations have unique naming restrictions within a domain. For more information, see “JMS Resource Naming Rules for Domain Interoperability” on page 9-3.

- Thresholds and quotas for messages and bytes (maximum number, and high and low thresholds), and whether or not bytes paging and/or messages paging is enabled on the destination.
- Message attributes that can be overridden, including priority, time-to-live, time-to-deliver, and delivery mode.
- Message redelivery attributes, including redelivery delay override, redelivery limit, and error destination.
- Multicasting attributes (for topics only), including multicast address, time-to-live (TTL), and port.

For instructions on creating and configuring a destination, see "JMS Destination Tasks" in the *Administration Console Online Help*.

Some destination attributes are dynamically configurable. When attributes are modified at run time, only incoming messages are affected; stored messages are not affected.

Configuring JMS Templates

A JMS template provides an efficient means of defining multiple destinations with similar attribute settings. JMS templates offer the following benefits:

- You do not need to re-enter every attribute setting each time you define a new destination; you can use the JMS template and override any setting to which you want to assign a new value.
- You can modify shared attribute settings dynamically simply by modifying the template.

To define the JMS template configuration attributes for destinations, use the Templates node in the Administration Console. The configurable attributes for a JMS template are the same as those configured for a destination. These configuration attributes are inherited by the destinations that use them, with the following exceptions:

- If the destination that is using a JMS template specifies an override value for an attribute, the override value is used.
- If the destination that is using a JMS template specifies a message redelivery value for an attribute, that redelivery value is used.

- The Name attribute is not inherited by the destination. This name is valid for the JMS template only. You must explicitly define a unique name for all destinations.

  Note: JMS templates must be uniquely named within a domain. For more information, see "JMS Resource Naming Rules for Domain Interoperability" on page 9-3.

- The JNDI Name, Enable Store, and Template attributes are not defined for JMS templates.
- The Multicast attributes are not defined for JMS templates because they apply only to topics.

Any attributes that are not explicitly defined for a destination are assigned default values. If no default value exists, be sure to specify a value within the JMS template or as a destination attribute override. If you do not do so, the configuration information remains incomplete, the WebLogic JMS configuration fails, and the WebLogic JMS does not boot.

For instructions on creating and configuring a JMS template, see "JMS Template Tasks" in the *Administration Console Online Help*.

Configuring Destination Keys

Use destination keys to define the sort order for a specific destination.

To create a destination key, use the Destination Keys node in the Administration Console and define the following configuration attributes:

- Name of the destination key
- Property name on which to sort
- Expected key type
- Direction in which to sort (ascending or descending)

For instructions on creating and configuring a destination key, see "Destination Key Tasks" in the *Administration Console Online Help*.

Configuring Stores

A persistent store consists of a file or database that is used for persistent messaging. To create a file or database store, use the Stores node in the Administration Console and define the following configuration attributes:

- Name of the JMS persistent store.
- For a JMS file store—provide the path to the location where the messages will be saved, and optionally select a disk write policy as explained in "Configuring a Synchronous Write Policy for JMS File Stores" on page 9-24.
- For a JMS JDBC database store—provide the JDBC connection pool and database table name prefix for use with multiple instances, as explained in "About JMS JDBC Store Table Prefixes" on page 9-18.

  Note: JMS stores must be uniquely named within a domain. For more information, see "JMS Resource Naming Rules for Domain Interoperability" on page 9-3.

Warning: You cannot configure a transaction (XA) connection pool to be used with a JDBC database store. For more information, see "JMS JDBC Transactions" on page 9-16.

JMS persistent stores can increase the amount of memory required during initialization of a WebLogic Server instance as the number of stored messages increases. If initialization fails due to insufficient memory while rebooting WebLogic Server, increase the heap size of the Java Virtual Machine (JVM) proportionally to the number of messages that are currently stored in the JMS persistent store. Then, try rebooting the server again. For more information on setting heap sizes, see "Tuning WebLogic Server Applications" in the *WebLogic Performance and Tuning Guide*.

For instructions on creating and configuring a store, see "JMS File Store Tasks" and "JMS JDBC Store Tasks" for information about file and JDBC database stores, respectively, in the *Administration Console Online Help*.

About JMS JDBC Stores

Through the use of JDBC, JMS enables you to store persistent messages in a database, which is accessed through a designated JDBC connection pool. The JMS database can be any database that is accessible through a JDBC driver. WebLogic JMS detects some drivers for the following databases:

- Pointbase
- Microsoft SQL (MSSQL) Server
- Oracle
- Sybase
- Cloudscape
- Informix
- IBM DB2
- Times Ten

The `weblogic/jms/ddl` directory within the `weblogic.jar` file contains JMS DDL files for these databases, which are actually text files containing the SQL commands that create the JMS database tables. To use a different database, simply copy and edit any one of these `.ddl` files.

Note: The JMS samples provided with your WebLogic Server distribution are set up to work with the Pointbase Java database. An evaluation version of Pointbase is included with WebLogic Server and a *demoPool* database is provided.

If your existing JMS JDBC stores somehow become corrupted, you can regenerate them using the `utils.Schema` utility. For more information see, "JDBC Database Utility" in *Programming WebLogic JMS*.

JMS JDBC Transactions

You cannot configure a transaction (XA) JDBC connection pool to be used with a JMS JDBC store. JMS must use a JDBC connection pool that uses a non-`XAResource` driver (you cannot use an XA driver or a JTS driver). JMS does the XA support above the JDBC driver.

This is because WebLogic JMS is its own resource manager. That is, JMS itself implements the `XAResource` and handles the transactions without depending on the database (even when the messages are stored in the database). This means that whenever you are using JMS and a database (even if it is the same database where the JMS messages are stored), then it is two-phase commit transaction. For more information about using transactions with WebLogic JMS, see "Using Transactions with WebLogic JMS" in *Programming WebLogic JMS*.

From a performance perspective, you may boost your performance if the JDBC connection pool used for the database work exists on the same WebLogic Server as the JMS queue—the transaction will still be two-phase, but it will be handled with less network overhead. Another performance boost might be achieved by using JMS file stores rather than JMS JDBC stores.

JMS JDBC Security

Optionally, you can restrict the JDBC connection pool resource. In previous releases of WebLogic Server, ACLs were used to protect WebLogic resources. In WebLogic Server version 7.0, security policies answer the question "who has access" to a WebLogic resource. A security policy is created when you define an association between a WebLogic resource and a user, group, or role. A WebLogic resource has no protection until you assign it a security policy. For instructions on how to set up security for all WebLogic Server resources, see "Setting Protections for WebLogic Resources" in the *Administration Console Online Help*.

Using Oracle Primary Keys with a JMS JDBC Store

For Oracle databases, Advanced Oracle Replication cannot replicate tables with the LONG or LONG RAW data type columns, but instead requires the use of Primary Keys. To achieve this, modify the default Oracle DDL file to manually add Primary Keys to the JMSStore table used by a JMS JDBC Store.

The `jms_oracle.ddl` file is preconfigured and supplied in the WebLogic `CLASSPATH` (in the `*WL_HOME*/server/lib/weblogic.jar` file, where `*WL_HOME*` is the top-level directory of your WebLogic Server installation).

For cases where the JMSStore and JMSState tables already exist, then these tables must be dropped and recreated to use a modified DDL file.

1. Use the JAR utility supplied with the JDK to extract the JMS Store DDL files to the `weblogic/jms/ddl` directory, using the following command:

```
jar xf weblogic.jar /weblogic/jms/ddl
```

2. Edit the `jms_oracle.ddl` text file as follows:

   a. Locate the `CREATE TABLE JMSSTORE` entry, and then add `PRIMARY KEY NOT NULL`, as follows:

   ```
   CREATE TABLE JMSStore (recordHandle int PRIMARY KEY NOT NULL,
   recordState int, record LONG RAW);
   ```

   b. Also, remove the following `CREATE INDEX` entry:

   ```
   CREATE INDEX JMSMSGQ_X ON JMSStore (recordHandle);
   ```

3. Save your changes to the DDL file.

4. Follow the "Regenerating JDBC Database Stores" instructions in *Programming WebLogic JMS* to regenerate the JMS Store tables that will be using your customized `jms_oracle.ddl` file.

About JMS JDBC Store Table Prefixes

The JMS database contains two system tables that are generated automatically and are used internally by JMS:

- `<`*`prefix`*`>JMSStore`
- `<`*`prefix`*`>JMSState`

The prefix name uniquely identifies JMS tables in the persistent store. Specifying unique prefixes allows multiple stores to exist in the same database. You configure the prefix via the Administration Console when configuring a JMS JDBC store. A prefix is prepended to table names when the DBMS requires fully qualified names, or when you must differentiate between JMS tables for two WebLogic Servers, enabling multiple tables to be stored on a single DBMS.

Warning: No two JMS JDBC stores should be allowed to use the same database tables, as this will result in data corruption.

Specify the prefix using the following format, which will result in a valid table name when prepended to the JMS table name:

```
[[[catalog.]schema.]prefix]JMSStore
```

where `catalog` identifies the set of system tables being referenced by the DBMS and `schema` translates to the ID of the table owner. For example, in a production database the JMS administrator could maintain a unique table for the Sales department, as follows:

```
[[[Production.]JMSAdmin.]Sales]JMSStore
```

The resulting table will be created in the `Production` catalog, under the `JMSAdmin` schema, and will be named `SalesJMSStore`.

Note: For some DBMS vendors, such as Oracle, there is no catalog to set or choose, so this format simplifies to `[[schema.]prefix]`. For more information, refer to your DBMS documentation for instructions on how to write and use a fully-qualified table name.

Recommended JDBC Connection Pool Settings for JMS JDBC Stores

WebLogic Server provides robust JDBC connection pools that can automatically reconnect to failed databases after they come back online, without requiring you to restart WebLogic Server. To take advantage of this capability, and make your use of JMS JDBC stores more robust, configure the following attributes on the JDBC connection pool associated with the JMS JDBC store:

```
TestConnectionsOnReserve=”true”
TestTableName=”SYSTABLES”
```

For more information about JDBC default Test Table Names, see "Connection Testing Options" in see “JDBC Connection Pool -> Configuration -> Testing” in the *Administration Console Online Help*.

Configuring Session Pools

Server session pools enable an application to process messages concurrently. After you define a JMS server, optionally, configure one or more session pools for each JMS server.

Use the Session Pools node in the Administration Console and define the following configuration attributes:

- Name of the server session pool. Session pools must be uniquely named within a domain. For more information, see "JMS Resource Naming Rules for Domain Interoperability" on page 9-3.
- Connection factory with which the server session pool is associated and is used to create sessions.
- Message listener class used to receive and process messages concurrently.
- Transaction attributes (acknowledge mode and whether or not the session pool creates transacted sessions).
- Maximum number of concurrent sessions.

For instructions on creating and configuring a session pool, see "Session Pool Tasks" in the *Administration Console Online Help*.

Some session pool attributes are dynamically configurable, but the new values do not take effect until the session pools are restarted.

Configuring Connection Consumers

Connection consumers are queues (Point-To-Point) or topics (Pub/Sub) that retrieve server sessions and process messages. After you define a session pool, configure one or more connection consumers for each session pool.

To configure connection consumers, use the Session Pools node in the Administration Console to define the following configuration attributes:

- Name of the connection consumer. Connection consumers must be uniquely named within a domain. For more information, see "JMS Resource Naming Rules for Domain Interoperability" on page 9-3.
- Maximum number of messages that can be accumulated by the connection consumer.
- JMS selector expression used to filter messages. For information about defining selectors, see Developing a WebLogic JMS Application" in *Programming WebLogic JMS*.
- Destination on which the connection consumer will listen.

To create and configure a connection consumer, and for detailed information about each of the connection consumer configuration attributes, see "Connection Consumer" in the *Administration Console Online Help*.

Monitoring JMS

Using the Administration Console, you can monitor statistics for the following JMS objects: JMS servers, connections, sessions, destinations, message producers, message consumers, server session pools, and durable subscribers.

JMS statistics continue to increment as long as the server is running. Statistics are reset only when the server is rebooted.

Note: For instructions on monitoring JMS connections to WebLogic Server, refer to the Servers section in the *Administration Console Online Help*.

Monitoring JMS Objects

To view run-time information for active JMS servers, destinations, and session pools:

1. Start the Administration Console.
2. Select the JMS node under Services, in the left pane, to expand the list of JMS servers.
3. Select the Server node under JMS in the left pane.

   The JMS server information is displayed in the right pane.
4. Select the JMS server that you want to monitor from the list or, from the JMS servers displayed in the right pane.
5. Select the Monitoring tab to display the following text links:
   - Monitor all Active JMS Servers
   - Monitor all Active JMS Destinations
   - Monitor all Active JMS Session Pool Runtimes

6. Click the appropriate text link to view monitoring data for that JMS object.

Note: When monitoring distributed destinations, you may see proxy topic members or system subscriptions for the topic or queue members. For more information see, "Monitoring Distributed Destination System Subscriptions and Proxy Topic Members" on page 9-22.

Monitoring Durable Subscribers

To view JMS durable subscribers that are running on destination topics:

1. Follow steps 1–3, as described in "Monitoring JMS Objects" on page 9-21.
2. Select the Destinations node under Servers in the left pane, to expand the list of JMS topic and queue destinations.

   The JMS destination information is displayed in a table format in the right pane, with the Durable Subscriber Runtimes column listing the number of durable subscribers running (if any) for the destination topics listed in the table.
3. To view durable subscriber information for a specific topic, click the icon (or actual number) in the Durable Subscriber Runtimes column for the desired topic.

Monitoring Distributed Destination System Subscriptions and Proxy Topic Members

In certain JMS configuration for Weblogic Server 7.0, distributed destinations may automatically create *proxy topic members* or *system subscriptions* between the topic or queue members. If this occurs, system subscriptions and proxy topic members will appear in MBean statistics, as well as in the Administration Console, when monitoring distributed destination members. They may also appear in the durable subscription names and in the consumer counts for the distributed destination members.

The following points describe the behavior of system subscriptions and proxy topic members:

- **Distributed Topic Proxy Members** — A WebLogic Server instance that has a configured JMS connection factory, but which has not been configured to host a local distributed topic member for a remote distributed topic, may automatically create and host a local proxy topic member for the remote distributed topic. This occurs when the first non-durable subscription for the distributed topic is created on the server's connection factory. The dynamically-created proxy topic member resides within a dynamically-created JMS server. Each of the manually-configured distributed topic members will create a system subscription for each dynamically-created proxy topic member. The non-durable consumers are then created on the proxy topic member.

- **Distributed Topic System Subscriptions** — System subscriptions are used to forward messages between configured distributed destination members. For example, when there are n members in a distributed topic, each member has at least n-1 system subscribers. In addition, for each proxy topic member, there will also be a system subscription on each distributed topic member.
- **Distributed Queue System Subscriptions** — Distributed queue members that have enabled the *Forward Delay* attribute on the distributed queue (by changing the default value of -1 seconds), may also create system subscribers. The system subscribers are used to forward messages from queue members with no consumers to queue members that do have consumers.
- **Durable System Subscriptions** — When a JMS file or JDBC store is configured for a distributed topic member, system subscriptions are created as durable subscribers. They are displayed by name in the Administration Console.

Tuning JMS

The following sections explain how to get the most out of your applications by implementing the administrative performance tuning features available with WebLogic JMS.

- Persistent Stores
- Using Message Paging
- Establishing Message Flow Control
- Tuning a Distributed Destination

Persistent Stores

The following sections describe the tuning options available when using persistent stores with WebLogic Server JMS.

Configuring a Synchronous Write Policy for JMS File Stores

By default, WebLogic JMS file stores guarantee up-to-the-message integrity by using synchronous writes. Disabling synchronous writes improves file store performance, often quite dramatically, but at the expense of possibly losing sent messages or generating duplicate received messages (even if messages are transactional) in the event of an operating system crash or a hardware failure. Simply shutting down an operating system will not generate these failures, as an OS flushes all outstanding writes during a normal shutdown. Instead, these failures can be emulated by shutting the power off to a busy server.

Note: The Synchronous Write Policy is ignored if the file store is used exclusively for paging non-persistent messages to disk.

Table 9-1 explains the options available when configuring a Synchronous Write Policy for all JMS file stores running on WebLogic Server.

Table 9-1 Synchronous Write Policy Attributes

Attribute	Description
Disabled	File store writes are allowed to use both the operating system's cache as well as the file system's on-disk cache. This policy is the fastest but the least reliable. It can be more than 100 times faster than the other policies, but power outages or operating system failures can cause lost and/or duplicate messages.
Cache-Flush	The default policy. Transactions cannot complete until all of their writes have been flushed down to disk. This policy is reliable and scales well as the number of simultaneous users increases.

Table 9-1 Synchronous Write Policy Attributes

Attribute	Description
Direct-Write	File store writes are written directly to disk. This policy is supported on Sun Solaris and Windows systems. If the Direct-Write policy is set on an unsupported platform, the file store automatically uses the Cache-Flush policy instead. The Direct-Write policy's reliability and performance depend on the platform's use of on-disk caches with respect to direct writes. For example, UNIX systems do not use on-disk caches for direct writes, while Windows systems generally do. The following points illustrate the pros and cons of using on-disk caching (when possible) with this policy: ■ With on-disk caching enabled, the Direct-Write policy can be 2-5 times faster than the Cache-Flush policy, except in highly scalable cases where it may be slightly slower. ■ With on-disk caching disabled, the Direct-Write policy is faster than the Cache-Flush policy in one-to-many cases, but much slower otherwise. ■ The Direct-Write policy scales well with on-disk caching enabled, but does not scale with it disabled. (Note that Sun Solaris does not allow enabling the on-disk cache for direct writes).

Warning: Unlike Sun Solaris, use of the Direct-Write policy on Windows may leave transaction data in the on-disk cache without writing it to disk immediately. This is not transactionally safe, as a power failure can cause loss of on-disk cache data, resulting in lost and/or duplicate messages. For reliable writes using Direct-Write on Windows, either disable all write caching for the disk (enabled by default), or use a disk with a battery backed cache.

To disable the on-disk cache for a hard disk on Windows, do the following: Start -> Settings -> Control Panel -> System -> Hardware tab -> click the Device Manager button -> Disk Drives -> double-click the *drive name* -> Disk Properties tab -> clear the Write Caching Enabled check box. Some file systems, however, do not allow this value to be changed (for example, a RAID system that has a reliable cache).

Comparing Policy Settings

The following tables compare the synchronous write policies with respect to reliability, performance, and scalability. Use the following key to interpret the expected results based on your synchronous write policy settings.

- **Disk Cache On/Off:** the on-disk write cache is enabled/disabled
- **1-m Perf:** very few clients
- **m-m Perf:** many clients
- **M-M Perf:** a large amount of concurrent clients
- **Reliability Low/High:** High reliability is needed for exactly-once (transactional) messaging

Table 9-2 Relative Performance (compare within same column)

Policy	Disk Cache	1-m Perf	m-m Perf	M-M Perf	Reliability
Disabled	On	****	****	****	Low (depends on OS cache)
	Off	****	****	****	Low
Cache-Flush	On	*	**	***	High
	Off	*	**	***	High
Direct-Write	On	**	***	**	Medium (High with reliable disk-cache
	Off	**	*	*	High

Table 9-3 Relative Scalability (compare within same row)

Policy	Disk Cache	1-m Perf	m-m Perf	M-M Perf
Disabled	On	****	****	****
	Off	****	****	****
Cache-Flush	On	*	**	***
	Off	*	**	***
Direct-Write	On	*	**	***
	Off	*	*	*

Using Message Paging

With the *message paging* feature, you can free up virtual memory during peak message load periods. This feature can greatly benefit applications with large message spaces.

JMS message paging saves memory for both persistent and non-persistent messages, as even persistent messages cache their data in memory. Paged persistent messages continue to be written to the regular backing store (file or database); and paged non-persistent messages are written to the JMS server's message paging store, which is configured separately.

A paged-out message does not free all of the memory that it consumes. The message header and message properties remain in memory for use with searching, sorting, and filtering.

Note: Messages sent in a transacted session are only eligible for paging *after* the session is committed. Prior to that, the message will only be held in memory; therefore, the heap size of the Java Virtual Machine (JVM) should be appropriately tuned to accommodate the projected peak amount of client load from all active sessions until they are committed. For more information on tuning your heap size, see "Tuning Java Virtual Machines (JVMs)" in *WebLogic Performance and Tuning*.

Configuring Paging

Unless paging is configured and enabled, all messages (even persistent ones) are held in memory. You can configure paging for a new or existing JMS server and/or its destinations through the Administration Console. Using the attributes on the JMS Server node you can specify a paging store for a JMS server, enable bytes and/or messages paging, and configure bytes/messages high and low thresholds to start and stop paging.

Similarly, using the attributes on the Destinations node, you can configure bytes/messages paging for all topics and queues configured on a JMS server. The destinations use the paging store that is configured for the JMS server.

Also, if you use JMS templates to configure multiple destinations, you can use the attributes on the Templates node to configure paging quickly on all your destinations. To override a template's paging configuration for specific destinations, you can enable or disable paging on any destination.

For instructions on configuring a new JMS server, templates, and destinations (Topics or Queues), see "JMS Server," "JMS Destination," and "JMS Template" in the *Administration Console Online Help*.

Note: For performance tuning purposes, you can modify the paging thresholds to any legal value at any time. Once paging is enabled, however, you cannot dynamically disable it by resetting a byte or message threshold back to -1. To prevent paging from occurring, set the byte/message high threshold to a very large number (maximum is 2^{63} -1), so that paging is not triggered.

Configuring a Paging Store for a JMS Server

Each JMS server must have its own paging store, which is used exclusively for paging out non-persistent messages for the JMS server and its destinations. It's best to use a JMS file store rather than a JMS JDBC store, as the JDBC store will perform poorly in comparison without any real benefit.

To configure a new paging store:

1. Start the Administration Console.
2. Click the JMS Store node. The right pane shows all the JMS stores.
3. Click the Create a new JMS File Store text link. The right pane shows the tabs associated with configuring a new file store.

4. Enter values in the attribute fields.
5. Click Create to create a file store instance with the name you specified in the Name field. The new instance is added under the JMS Stores node in the left pane.
6. If you have multiple JMS servers in your domain, repeat steps 3-5 for each server instance.

Configuring Paging on a JMS Server

To enable and configure paging on an existing JMS server:

1. Click the JMS Servers node. The right pane shows all the servers defined in your domain.
2. Click the server that you want to configure for paging. The right pane shows the tabs associated with configuring the server.
3. On the General tab, use the Paging Store list box to select the store that you configured to store the paged messages. Click Apply to save your changes.

   For instructions on configuring a paging store, refer to "Configuring a Paging Store for a JMS Server" on page 9-28.
4. On the Thresholds & Quotas tab, configure bytes paging:
   - Select the Bytes Paging Enabled check box.
   - In the Bytes Threshold High field, enter an amount that will start bytes paging when the number of bytes on the JMS server exceeds this threshold.
   - In the Bytes Threshold Low field, enter an amount that will stop bytes paging once the number of bytes on the JMS server falls below this threshold.
5. On the Thresholds & Quotas tab, configure messages paging:
   - Select the Messages Paging Enabled check box.
   - In the Messages Threshold High field, enter an amount that will start messages paging when the number of messages on the JMS server exceeds this threshold.

- In the Messages Threshold Low field, enter an amount that will stop messages paging once the number of messages on the JMS server falls below this threshold.

6. Click Apply to save the new bytes and/or messages paging values.
7. Repeat steps 2–6 to configure paging for additional JMS servers in the domain.

   Note: Each JMS server must have its own paging store.

8. After you configure your JMS server (or servers) for paging, do one of the following:
   - If you are not configuring a JMS server's destinations for paging, reboot WebLogic Server to activate paging.
   - If you want to configure a server's destinations for paging, follow refer to either "Configuring Paging on a JMS Template" on page 9-30 or "Configuring Paging on Destinations" on page 9-31.

Configuring Paging on a JMS Template

JMS templates provide an efficient way to define multiple destinations (topics or queues) with similar attribute settings. To configure paging on a template for destinations, do the following:

1. Click the JMS node in the left pane.
2. Click the JMS Templates node. The right pane shows all the templates defined in the domain.
3. Click the template that you want to configure for paging. The right pane shows the tabs associated with configuring the template.
4. On the Thresholds & Quotas tab, configure bytes paging:
   - Select the Bytes Paging Enabled check box.
   - In the Bytes Threshold High field, enter an amount that will start bytes paging when the number of bytes on the JMS server exceeds this threshold.
   - In the Bytes Threshold Low field, enter an amount that will stop bytes paging once the number of bytes on the JMS server falls below this threshold.
5. On the Thresholds & Quotas tab, configure messages paging:

- Select the Messages Paging Enabled check box.
- In the Messages Threshold High field, enter an amount that will start messages paging when the number of messages on the JMS server exceeds this threshold.
- In the Messages Threshold Low field, enter an amount that will stop messages paging once the number of messages on the JMS server falls below this threshold.

6. Click Apply to save the new bytes and/or messages paging values.
7. Repeat steps 3–6 to configure paging for additional JMS templates.
8. After configuring all of your JMS templates for paging, reboot WebLogic Server to activate paging.

Configuring Paging on Destinations

Follow these directions if you are configuring paging on destinations without using a JMS template.

1. Under JMS Servers, expand a server instance that is already configured for paging.
2. Click the Destinations node. The right pane shows all of the server's topics and queues.
3. Click the topic or queue that you want to configure for paging. The right pane shows the tabs associated with configuring the topic or queue.
4. On the Thresholds & Quotas tab, configure bytes paging:
    - Select the Bytes Paging Enabled check box.
    - In the Bytes Threshold High field, enter an amount that will start bytes paging when the number of bytes on the JMS server exceeds this threshold.
    - In the Bytes Threshold Low field, enter an amount that will stop bytes paging once the number of bytes on the JMS server falls below this threshold.
5. On the Thresholds & Quotas tab, configure messages paging:
    - Select the Messages Paging Enabled check box.

- In the Messages Threshold High field, enter an amount that will start messages paging when the number of messages on the JMS server exceeds this threshold.
- In the Messages Threshold Low field, enter an amount that will stop messages paging once the number of messages on the JMS server falls below this threshold.

6. Click Apply to save the new bytes and/or messages paging values.
7. Repeat steps 3–6 to configure paging for additional JMS destinations.
8. After you configure all your destinations for paging, reboot WebLogic Server to activate paging.

Note: If you use JMS templates to configure your destinations, a destination's explicit Byte/Messages Paging configuration overrides the template's configuration. For more information, refer to "Configuring a Destination to Override Paging on a JMS Template" on page 9-32 and to "Configuring JMS" on page 9-2.

Configuring a Destination to Override Paging on a JMS Template

Follow these directions if you want to override a template's settings and enable or disable paging on a specific destination.

1. Under JMS Servers, expand a server instance that is already configured for paging.
2. Click the Destinations node. The right pane shows all of the server's topics and queues.
3. Click the topic or queue that you want to configure for paging. The right pane shows the topics or queues associated with the server instance.
4. On the Thresholds & Quotas tab, configure the Bytes Paging Enabled and/or Messages Paging Enabled attributes on the destination according to how you want to override the JMS template for the destination.
    - To disable paging for the destination, select False in the Bytes Paging Enabled and/or the Messages Paging Enabled list boxes.
    - To enable paging for the destination, select True in the Bytes Paging Enabled and/or the Messages Paging Enabled list boxes.
5. Click Apply to save the new bytes and/or messages paging values.

6. Repeat steps 2–5 to configure paging for additional JMS destinations on the same server instance.
7. Once all of your destinations are configured for paging, then reboot WebLogic Server to activate paging.

JMS Paging Attributes

The following sections briefly describe the paging attributes available with WebLogic Server JMS.

JMS Server Paging Attributes

Table 9-4 describes the paging attributes that you define when configuring paging on a JMS Server. For detailed information about other JMS Server attributes, and the valid and default values for them, see "JMS Server" in the *Administration Console Online Help*.

Table 9-4 JMS Server Attributes

Attribute	Description
Bytes Paging Enabled	■ If the Bytes Paging Enabled check box is not selected (False), then server bytes paging is explicitly disabled. ■ If the Bytes Paging Enabled check box is selected (True), a paging store has been configured, and both the Bytes Threshold Low and Bytes Threshold High attributes are greater than -1, then server bytes paging is enabled. ■ If either the Bytes Threshold Low or Bytes Threshold High attribute is undefined, or defined as -1, then server bytes paging is implicitly disabled—even though the Bytes Paging Enabled check box is selected (True).

Table 9-4 JMS Server Attributes

Attribute	Description
Messages Paging Enabled	■ If the Messages Paging Enabled check box is not selected (False), then server messages paging is explicitly disabled. ■ If the Messages Paging Enabled check box is selected (True), a paging store has been configured, and both the Messages Threshold Low and Messages Threshold High attributes are greater than -1, then server messages paging is enabled. ■ If either the Messages Threshold Low or Messages Threshold High attribute is undefined, or defined as -1, then server paging is implicitly disabled—even though the Messages Paging Enabled check box is selected (True).
Paging Store	The name of the persistent store where non-persistent messages are paged. A paging store cannot be the same store used for persistent messages or durable subscribers. Two JMS servers cannot use the same paging store; therefore, you must configure a unique paging store for each server.

JMS Template Paging Attributes

Table 9-5 describes the paging attributes that you define when configuring paging on JMS templates for destinations. For detailed information about other JMS template attributes, and the valid and default values for them, see "JMS Template" in the *Administration Console Online Help*.

Table 9-5 JMS Template Attributes

Attribute	Description
Bytes Paging Enabled	■ If the Bytes Paging Enabled check box is not selected (False), then destination-level bytes paging is disabled for the JMS template's destinations—unless the destination setting overrides the template. ■ If the Bytes Paging Enabled check box is selected (True), a paging store has been configured for the JMS Server, and both the Bytes Threshold Low and Bytes Threshold High attributes are greater than -1, then destination-level bytes paging is enabled for the JMS template's destinations—unless the destination setting overrides the template. ■ If no value is defined in the JMS Template MBean, then the value defaults to False and bytes paging is disabled for the JMS template's destinations.
Messages Paging Enabled	■ If the Messages Paging Enabled check box is not selected (False), then destination-level messages paging is disabled for the template's destination—unless the destination setting overrides the template. ■ If the Messages Paging Enabled check box is selected (True), a paging store has been configured for the JMS Server, and both the Messages Threshold Low and Messages Threshold High attributes are greater than -1, then destination-level messages paging is enabled for this destination—unless the destination setting overrides the template. ■ If no value is defined in the JMS Template MBean, then the value defaults to False and messages paging is disabled for the template's destinations.

JMS Destination Paging Attributes

Table 9-6 describes the attributes that you define when configuring paging on destinations. For detailed information about other JMS destination attributes, and valid and default values for them, see "JMS Destination" in the *Administration Console Online Help*.

Table 9-6 JMS Destination Attributes

Attribute	Description
Bytes Paging Enabled	■ If Bytes Paging Enabled is set to False, then destination-level bytes paging is disabled for this destination. ■ If Bytes Paging Enabled is set to True, a paging store has been configured for the JMS Server, and both the Bytes Threshold Low and Bytes Threshold High attributes are greater than -1, then destination-level bytes paging is enabled for this destination. ■ If Bytes Paging Enabled is set to Default, then this value inherits the template's value—if a template is specified. If no template is configured for the destination, then the Default value is equivalent to False.
Messages Paging Enabled	■ If Messages Paging Enabled is set to False, then destination-level messages paging is disabled for this destination. ■ If Messages Paging Enabled is set to True, a paging store has been configured for the JMS Server, and both the Messages Threshold Low and Messages Threshold High attributes are greater than -1, then destination-level messages paging is enabled for this destination. ■ If Messages Paging Enabled is set to Default, then this value inherits the template's value—if a template is specified. If no template is configured for the destination, then the Default value is equivalent to False.

Note: If server paging is enabled, and destination-level paging is disabled for a given destination, than messages on the destination can still be paged if server paging is triggered. However, when destination-level paging is disabled for a given destination, then the destination's high thresholds will not force the destination to page out messages when they are exceeded.

Paging Threshold Attributes

Table 9-7 briefly describes the bytes and messages paging thresholds available with JMS servers, templates, and destinations. For detailed information about other JMS server, template, and destination attributes, and the valid and default values for them, see "JMS Server," "JMS Destination," and "JMS Template" in the *Administration Console Online Help*.

Table 9-7 Paging Threshold Attributes

Attribute	Description
Bytes Threshold High	Start paging when the number of bytes exceeds this threshold.
Bytes Threshold Low	Stop paging when the number of bytes falls back below this threshold.
Messages Threshold High	Start paging when the number of messages exceeds this threshold.
Messages Threshold Low	Stop paging when the number of messages falls back below this threshold.

The thresholds are defined for servers, templates, and destinations as follows:

- If either bytes high/low threshold value is not defined (or is defined as -1), then the number of bytes is not used to determine when and what to page.
- If either messages high/low threshold value is not defined (or is defined as -1), then the number of messages is not used to determine when and what to page.
- A server or template/destination must have the Bytes/Messages Paging Enabled attribute set to True in order for paging to take place. If the thresholds are set, but paging is not enabled, messages are still logged on the server indicating threshold conditions.

Establishing Message Flow Control

With the WebLogic JMS flow control feature, you can enable a JMS server or destination to slow down message producers when it determines that it is becoming overloaded. Specifically, when a JMS server/destination exceeds its specified bytes or messages thresholds, it becomes *armed* and instructs producers to limit their message flow (messages per second).

Producers limit their production rate based on a set of flow control attributes configured for producers via the JMS connection factory. Starting at a specified *flow maximum* number of messages, a producer evaluates whether the server/destination is still armed at prescribed intervals (for example, every 10 seconds for 60 seconds). If at each interval, the server/destination is still armed, then the producer continues to move its rate down to its prescribed *flow minimum* amount.

As producers slow themselves down, the threshold condition gradually corrects itself until the server/destination is *unarmed*. At this point, a producer is allowed to increase its production rate, but not necessarily to the maximum possible rate. In fact, its message flow continues to be controlled (even though the server/destination is no longer armed) until it reaches its prescribed *flow maximum*, at which point it is no longer flow controlled.

Configuring Flow Control

Producers receive a set of flow control attributes from their session, which receives the attributes from the connection, and which receives the attributes from the connection factory. You configure the flow control attributes on the JMS connection factory via the Administration Console.

These attributes allow the producer to adjust its message flow. Specifically, the producer receives attributes that limit its flow within a minimum and maximum range. As conditions worsen, the producer moves toward the minimum; as conditions improve; the producer moves toward the maximum. Movement toward the minimum and maximum are defined by two additional attributes that specify the rate of movement toward the minimum and maximum. Also, the need for movement toward the minimum and maximum is evaluated at a configured interval.

To configure message flow control on a connection factory, follow these steps:

1. Expand the JMS node.

2. Click the Connection Factories node. The JMS Connection Factories table displays in the right pane showing all the connection factories defined in your domain.
3. Click the connection factory you want to establish message flow control for. A dialog displays in the right pane showing the tabs associated with modifying a connection factory.
4. On the Flow Control tab, define the attributes as described in following table:

Table 9-8 Flow Control Attributes

Attribute	Description
Flow Control Enabled	Determines whether a producer can be flow controlled by the JMS server.
Flow Maximum	The maximum number of messages per second for a producer that is experiencing a threshold condition. If a producer is not currently limiting its flow when a threshold condition is reached, the initial flow limit for that producer is set to Flow Maximum. If a producer is already limiting its flow when a threshold condition is reached (the flow limit is less than Flow Maximum), then the producer will continue at its current flow limit until the next time the flow is evaluated. Once a threshold condition has subsided, the producer is not permitted to ignore its flow limit. If its flow limit is less than the Flow Maximum, then the producer must gradually increase its flow to the Flow Maximum each time the flow is evaluated. When the producer finally reaches the Flow Maximum, it can then ignore its flow limit and send without limiting its flow.
Flow Minimum	The minimum number of messages per second for a producer that is experiencing a threshold condition. This is the lower boundary of a producer’s flow limit. That is, WebLogic JMS will not further slow down a producer whose message flow limit is at its Flow Minimum.
Flow Interval	An adjustment period of time, defined in seconds, when a producer adjusts its flow from the Flow Maximum number of messages to the Flow Minimum amount, or vice versa.

Table 9-8 Flow Control Attributes

Attribute	Description
Flow Steps	The number of steps used when a producer is adjusting its flow from the Flow Minimum amount of messages to the Flow Maximum amount, or vice versa. Specifically, the Flow Interval adjustment period is divided into the number of Flow Steps (for example, 60 seconds divided by 6 steps is 10 seconds per step). Also, the movement (that is, the rate of adjustment) is calculated by dividing the difference between the Flow Maximum and the Flow Minimum into steps. At each Flow Step, the flow is adjusted upward or downward, as necessary, based on the current conditions, as follows: ■ The downward movement (the decay) is geometric over the specified period of time (Flow Interval) and according to the specified number of Flow Steps. (For example, 100, 50, 25, 12.5). ■ The movement upward is linear. The difference is simply divided by the number of steps.

5. Click Apply to store new attribute values.

For detailed information about other connection factory attributes, and the valid and default values for them, see "JMS Connection Factory" in the *Administration Console Online Help*.

Flow Control Thresholds

The attributes used for configuring bytes/messages thresholds are defined as part of the JMS server and JMS destination. Table 9-9 defines how these thresholds start and stop flow control on a JMS server and/or JMS destination.

Table 9-9 Flow Control Threshold Attributes

Attribute	Description
Bytes/Messages Threshold High	When the number of bytes/messages exceeds this threshold, the JMS server/destination becomes armed and instructs producers to limit their message flow.

Table 9-9 Flow Control Threshold Attributes

Attribute	Description
Bytes/Messages Threshold Low	When the number of bytes/messages falls below this threshold, the JMS server/destination becomes unarmed and instructs producers to begin increasing their message flow. Flow control is still in effect for producers that are below their message flow maximum. Producers can move their rate upward until they reach their flow maximum, at which point they are no longer flow controlled.

For detailed information about other JMS server and JMS destination attributes, and the valid and default values for them, see "JMS Server" and "JMS Destination" in the *Administration Console Online Help*.

Tuning a Distributed Destination

The following sections describe how to tune a distributed destination by configuring the Load Balancing and Server Affinity attributes on the JMS Connection Factory. For more information about configuring a distributed destination, see "Configuring Distributed Destinations" on page 9-44.

Configuring Message Load Balancing for a Distributed Destination

The Load Balancing Enabled attribute on the JMS Connection Factory → Configuration →General tab defines whether non-anonymous producers created through a connection factory are load balanced on a per-call basis. Applications that use distributed destinations to distribute or balance their producers and consumers across multiple physical destinations, but do not want to make a load balancing decision each time a message is produced, can turn off the Load Balancing Enabled attribute.

To ensure a fair distribution of the messaging load among a distributed destination, the initial physical destination (queue or topic) used by producers is always chosen at random from among the distributed destination members.

To configure load balancing on a connection factory:

1. Expand the JMS node.
2. Click the Connection Factories node. The JMS Connection Factories table displays in the right pane showing all the connection factories defined in your domain.
3. Click the connection factory on which you want to establish message load balancing. A dialog displays in the right pane showing the tabs associated with modifying a connection factory.
4. Define the Load Balancing Enabled attribute in the General tab.
   - *Load Balancing Enabled = True*
     For `Queue.sender.send()` methods, non-anonymous producers are load balanced on every invocation across the distributed queue members.

     For `TopicPublish.publish()` methods, non-anonymous producers are *always* pinned to the same physical topic for every invocation, irrespective of the Load Balancing Enabled setting.
   - *Load Balancing Enabled = False*
     Producers always produce to the same physical destination until they fail. At that point, a new physical destination is chosen.
5. Click Apply to save your changes.

   Note: Depending on your implementation, the setting of the Server Affinity Enabled attribute can affect the load balancing preferences for distributed destinations. For more information, see "How Distributed Destination Load Balancing Is Affected When Using the Server Affinity Enabled Attribute" in *Programming WebLogic JMS*.

Anonymous producers (producers that do not designate a destination when created), are load-balanced each time they switch destinations. If they continue to use the same destination, then the rules for non-anonymous producers apply (as stated previously).

For more information about how message load balancing takes place among the members of a distributed destination, see "Load Balancing Messages Across a Distributed Destination" in *Programming WebLogic JMS*.

Configuring Server Affinity for a Distributed Destination

The Server Affinity Enabled attribute on the JMS Connection Factory →Configuration →General tab defines whether a WebLogic Server that is load balancing consumers or producers across multiple physical destinations in a distributed destination set, will first attempt to load balance across any other physical destinations that are also running on the same WebLogic Server.

Note: The Server Affinity Enabled attribute does not affect queue browsers. Therefore, a queue browser created on a distributed queue can be pinned to a remote distributed queue member even when Server Affinity is enabled.

To disable server affinity on a connection factory:

1. Expand the JMS node.
2. Click the Connection Factories node. The JMS Connection Factories table displays in the right pane showing all the connection factories defined in your domain.
3. Click the connection factory on which you want to disable server affinity. A dialog displays in the right pane showing the tabs associated with modifying a connection factory.
4. Define the Server Affinity Enabled attribute in the General tab.
   - *Server Affinity Enabled = Tru*e
     A WebLogic Server instance that is load balancing consumers or producers across multiple physical destinations in a distributed destination set, will first attempt to load balance across any other physical destinations that are also running on the same WebLogic Server.
   - *Server Affinity Enabled = False*
     A WebLogic Server instance will load balance consumers or producers across physical destinations in a distributed destination set and disregard any other physical destinations also running on the same WebLogic Server.
5. Click Apply to save your changes.

For more information about how the Server Affinity Enabled setting affects the load balancing among the members of a distributed destination, see "How Distributed Destination Load Balancing Is Affected When Using the Server Affinity Enabled Attribute" in *Programming WebLogic JMS*.

Configuring Distributed Destinations

A *distributed destination* is a set of physical destinations (queues or topics) that are called under a single JNDI name so they appear to be a single, logical destination to a client when the members of the set are actually distributed across multiple servers within a cluster, with each destination member belonging to a separate JMS server.

By enabling you to configure multiple physical queues and topics as members of a distributed destination, WebLogic JMS supports high availability and load balancing of the JMS destinations within a cluster. For instructions on using a distributed destination with your applications, see "Developing a WebLogic JMS Application" in *Programming WebLogic JMS*.

Guidelines for Configuring Distributed Destinations

You configure distributed JMS destinations through the Services →JMS →Distributed Destinations node. To facilitate the configuration process, these instructions are divided into procedures that address the following scenarios:

- New implementations of WebLogic JMS with no physical destinations *or* existing configurations of WebLogic JMS that do not require previously configured destinations to be part of a distributed destination:
  - Creating a Distributed Topic and Creating Members Automatically
  - Creating a Distributed Queue and Creating Members Automatically
- Existing implementations of WebLogic JMS that require previously configured destinations to be members of a distributed destination set:
  - Creating a Distributed Topic and Adding Existing Physical Topics as Members Manually
  - Creating a Distributed Queue and Adding Existing Physical Queues as Members Manually

Configuration Best Practices for Distributed Destinations

To simplify the management and development of distributed destination applications, BEA recommends deploying a distributed destination consistently across the cluster by using similar configurations for each participating JMS server and it's associated physical destinations. This should include consistent configuration of JMS stores, message paging, error destinations, message/bytes quotas, and connection factories. For example, when persistent messaging is configured, you should ensure that each distributed member is associated with a similar JMS store, with regards to capacity and performance.

The recommended method for maintaining configuration consistency for distributed destination members is to use JMS templates, since they are the most reliable way to centralize any custom configuration that is common to each physical destination. For newly created distributed destination members configured using the Administration Console, use the JMS template that is automatically created for those members, as described in "Automatic JMS Template Creation" on page 9-46. And for a distributed destination created to represent existing physical destinations, use the JMS template associated with those physical destinations if one exists; otherwise, configure one, as described in "Configuring JMS Templates" on page 9-13.

Warning: To avoid application failure issues, consistent security permissions *must* be configured on *each* participating distributed destination member. For example, if one member of a distributed destination has different send restrictions than the other members, an application may sometimes be allowed to send to the distributed destination, and yet other times it may be forbidden.

Load Balancing and Server Affinity Tuning

The default Load Balancing Enabled and Server Affinity Enabled values for tuning a distributed destination configuration can be modified on the JMS connection factory through the Administration Console. For more information, see "Configuring Message Load Balancing for a Distributed Destination" on page 9-41 and "Configuring Server Affinity for a Distributed Destination" on page 9-43.

Automatic JMS Template Creation

When a distributed destination is created with new distributed topic or queue members through the Administration Console, a corresponding JMS template is automatically created with default attribute values for those members. The new JMS template appears under the JMS Templates node with the same name as the distributed destination. The thresholds, quotas, and other attributes for the distributed destination members can be reset using this template. For more information on using JMS templates, see "Configuring JMS Templates" on page 9-13.

JMS Server Removal Precaution

In a clustered configuration, untargeting or deleting a JMS server that is hosting a distributed destination member — without first manually deleting that member from the distributed destination configuration — can cause the configuration file (`config.xml`) to become unbootable. For instructions on deleting distributed destination members, see "Deleting a JMS Distributed Queue Member" on page 9-57 and "Deleting a JMS Distributed Topic Member" on page 9-59.

Creating a Distributed Topic and Creating Members Automatically

Follow these steps to configure a distributed topic and automatically create its topic members on JMS servers that are part of a WebLogic Server cluster (for high availability) or on an single WebLogic Server instance that is not part of a cluster.

1. Expand the JMS node and the Distributed Destinations node.
2. Click the Configure a new Distributed Topic link in the right pane. A Configuration dialog shows the tabs associated with configuring a new distributed topic.
3. Define the attributes in the General tab according to the following table.

Table 9-10 Distributed Topic Attributes on the General Tab

Attribute	Description
Name	Uniquely identify the distributed topic within a WebLogic Server domain.
JNDI Name	Enter the name used to bind the distributed topic into the JNDI namespace. Applications use the JNDI Name to look up the distributed topic. A distributed topic that does not have a JNDI Name can be referenced by passing the Name of the distributed destination to: `javax.jms.TopicSession.createTopic()`.
Load Balancing Policy	Define how producers will distribute their messages across the members of a distributed topic. The valid values are Round-Robin and Random as defined in "Configuring Message Load Balancing for a Distributed Destination" on page 9-41.

4. Click Create to create the distributed topic.

5. On the Thresholds & Quotas tab, define the following attributes for all of the distributed topic members:

   - Thresholds and quotas for messages and bytes (maximum number and high/low thresholds).

   - Whether or not bytes paging and/or messages paging is enabled on the distributed topic.

   For more information about these attributes, see "JMS Distributed Topic -> Configuration -> Thresholds & Quotas" in the Administration Console Online Help.

6. Click Apply to store new attribute values.

7. On the Auto Deploy tab, indicate the WebLogic Server instances where you want the distributed topic members to be automatically created.

8. Click the Create members on the selected Servers (and JMS Servers) text link. An auto deploy dialog prompts you to select one of the following options:

   - Select a cluster at which to target the distributed topic, and then click Next.

or

- Accept the None option to bypass this dialog so you can select an individual server or servers that are part of the cluster. (In this case, skip to Step 10).

9. If you selected a cluster, do the following to select WebLogic Server instances within the cluster:

    a. All WebLogic Server instances that are members of the cluster and that are not already hosting a distributed topic are listed and are selected by default. To exclude a server instance from hosting the distributed topic, clear the corresponding check box.

    b. Click Next to proceed to the next dialog.

    c. Skip to Step 11 to select the JMS servers that are available on the selected WebLogic Servers for creating distributed topic members.

10. If you selected None on the Cluster dialog in Step 8, select a single WebLogic Server instance in the domain:

    a. From the list box, select an individual server where you want to create a distributed topic member.

    b. Click Next to proceed to the next dialog.

11. All JMS servers that are deployed on the selected WebLogic Server instances, and that are not already hosting a distributed topic, are listed and are selected by default. To exclude a JMS server from hosting the distributed topic member, clear the corresponding check box.

    If there are no existing distributed topic members on the selected JMS servers, one new JMS Topic will be created on each JMS server and added as a member of the distributed topic.

12. Click Next to proceed to the final Auto Deploy dialog.

13. Click Apply to save your Auto Deploy selections.

14. Click the Configuration --> Members tab to view the topic members that were automatically created for the new distributed topic.

15. Expand the JMS →Templates node to view the JMS template that was automatically created with the same name as the distributed topic.

Creating a Distributed Topic and Adding Existing Physical Topics as Members Manually

For existing implementations of WebLogic JMS that require previously configured destinations to be members of a distributed destination set, follow these steps to configure a distributed topic and manually add your existing physical topics as members:

1. Expand the JMS node and the Distributed Destinations node.
2. Click the Configure a new Distributed Topic link in the right pane. A Configuration dialog shows the tabs associated with configuring a new distributed topic.
3. Define the attributes in the General tab according to the following table.

Table 9-11 Distributed Topic Attributes on the General Tab

Attribute	Description
Name	Uniquely identify the distributed topic within a WebLogic Server domain.
JNDI Name	Enter the name used to bind the distributed topic into the JNDI namespace. Applications use the JNDI Name to look up the distributed topic. A distributed topic that does not have a JNDI Name can be referenced by passing the Name of the distributed destination to: `javax.jms.TopicSession.createTopic()`.
Load Balancing Policy	Define how producers will distribute their messages across the members of a distributed topic. The valid values are Round-Robin and Random as defined in "Configuring Message Load Balancing for a Distributed Destination" on page 9-41.

4. Click Create to create the distributed topic.
5. On the Thresholds & Quotas tab, define the following attributes for all of the distributed topic members:

- Thresholds and quotas for messages and bytes (maximum number and high/low thresholds).
- Whether or not bytes paging and/or messages paging is enabled on the distributed topic.

If a distributed topic member's underlying physical topic already has a JMS Template with configured thresholds and quotas, these attributes do not apply to that topic member. For more information about these attributes, see "JMS Distributed Topic -> Configuration -> Thresholds & Quotas" in the Administration Console Online Help.

6. Click Apply to store new attribute values.

   Note: If you want to automatically create topic members on JMS servers that are part of a WebLogic Server cluster (for high availability) or on an single WebLogic Server instance that is not part of a cluster, see "Creating a Distributed Topic and Creating Members Automatically" on page 9-46.

7. On the Configuration -->Members tab, create distributed topic members for your existing physical topics.
8. Click the Configure a new Distributed Topic Member link in the right pane. A Configuration dialog shows the tabs associated with configuring a new distributed topic member.
9. Define the attributes in the General tab according to the following table.

Table 9-12 Distributed Topic Member Attributes on the General Tab

Attribute	Description
Name	Uniquely identify the distributed topic member within a WebLogic Server domain.
JMSTopic	Select the underlying physical topic that is associated with the distributed topic member.

Table 9-12 Distributed Topic Member Attributes on the General Tab

Attribute	Description
Weight	Define the weight (that is, a measure of ability to handle message load) of the topic member with respect to other topic members in the distributed destination. The random distribution load-balancing algorithm uses the weight assigned to the physical destinations to compute a weighted distribution for the set of physical destinations. For more information, see "Developing a WebLogic JMS Application" in *Programming WebLogic JMS*.

10. Click Create to create the new distributed topic member. The new member is added to the Distributed Topic table.
11. If necessary, repeat steps 8–10 to continue adding topic members to the distributed topic.

Creating a Distributed Queue and Creating Members Automatically

Follow these steps to configure a distributed queue and automatically create its queue members on JMS servers that are part of a WebLogic Server cluster (for high availability) or on an single WebLogic Server instance that is not part of a cluster.

1. Expand the JMS node and the Distributed Destinations node.
2. Click the Configure a new Distributed Queue link in the right pane. A Configuration dialog shows the tabs associated with configuring a new distributed queue.
3. Define the attributes in the General tab according to the following table.

Table 9-13 Distributed Queue Attributes on the General Tab

Attribute	Description
Name	Uniquely identify the distributed queue within a WebLogic Server domain.

Table 9-13 Distributed Queue Attributes on the General Tab

Attribute	Description
JNDI Name	Enter the name used to bind the distributed queue into the JNDI namespace. Applications use the JNDI Name to look up the distributed queue. A distributed queue that does not have a JNDI Name can be referenced by passing the Name of the distributed destination to `javax.jms.QueueSession.createQueue()`.
Load Balancing Policy	Define how producers will distribute their messages across the members of a distributed queue. The valid values are Round-Robin and Random as defined in "Configuring Message Load Balancing for a Distributed Destination" on page 9-41.
Forward Delay	Define the amount of time, in seconds, that a distributed queue member with messages, but which has no consumers, will wait before forwarding its messages to other queue members that do have consumers.

4. Click Create to create the distributed queue.
5. On the Thresholds & Quotas tab, define the following attributes for all of the distributed queue members:
   - Thresholds and quotas for messages and bytes (maximum number and high/low thresholds).
   - Whether or not bytes paging and/or messages paging is enabled on the distributed queue.

   For more information about these attributes, see "JMS Distributed Queue -> Configuration -> Thresholds & Quotas" in the Administration Console Online Help.
6. Click Apply to store new attribute values.
7. On the Auto Deploy tab, indicate the WebLogic Server instances where you want the distributed queue members to be automatically created.
8. Click the Create members on the selected Servers (and JMS Servers) text link. A dialog prompts you to select one of the following options:

- Select a cluster at which to target the distributed queue, and then click Next.

  or

- Accept the None option to bypass this dialog so you can select an individual server that is not in a cluster. (In this case, skip to Step 10).

9. If you selected a cluster, do the following to select WebLogic Server instances within the cluster:

   a. All servers that are members of the cluster, and which are not already hosting a distributed queue, are listed and are selected by default. To exclude a server from hosting the distributed queue, clear the corresponding check box.

   b. Click Next to proceed to the next dialog.

   c. Skip to Step 11 to select the JMS servers that are available on the selected WebLogic Servers for creating distributed queue members.

10. If you selected None on the Cluster dialog in Step 8, select a single WebLogic Server instance in the domain:

    a. From the list box, select an individual server where you want to create the distributed queue member.

    b. Click Next to proceed to the next dialog.

11. All JMS servers that are deployed on the selected WebLogic Servers, and which are not already hosting a distributed queue, are listed and are selected by default. To exclude a JMS server from hosting the distributed queue member, clear the corresponding check box.

    If there are no existing distributed queue members on the selected JMS servers, one new JMS Queue will be created on each JMS server and added as a member of the distributed queue.

12. Click Next to proceed to the final Auto Deploy dialog.

13. Click Apply to save your Auto Deploy selections.

14. Click the Configuration --> Members tab to view the queue members that were automatically created for the new distributed queue.

15. Expand the JMS →Templates node to view the JMS template that was automatically created with the same name as the distributed queue.

Creating a Distributed Queue and Adding Existing Physical Queues as Members Manually

For existing implementations of WebLogic JMS that require previously configured destinations to be members of a distributed destination set, follow these steps to configure a distributed queue and manually add your existing physical queues as members.

1. Expand the JMS node and the Distributed Destinations node.
2. Click the Configure a new Distributed Queue link in the right pane. A Configuration dialog shows the tabs associated with configuring a new distributed queue.
3. Define the attributes in the General tab according to the following table.

Table 9-14 Distributed Queue Attributes on the General Tab

Attribute	Description
Name	Uniquely identify the distributed queue within a WebLogic Server domain.
JNDI Name	Enter the name used to bind the distributed queue into the JNDI namespace. Applications use the JNDI Name to look up the distributed queue. A distributed queue that does not have a JNDI Name can be referenced by passing the Name of the distributed destination to `javax.jms.QueueSession.createQueue()`.
Load Balancing Policy	Define how producers will distribute their messages across the members of a distributed queue. The valid values are Round-Robin and Random as defined in "Configuring Message Load Balancing for a Distributed Destination" on page 9-41.
Forward Delay	Define the amount of time, in seconds, that a distributed queue member with messages, but which has no consumers, will wait before forwarding its messages to other queue members that do have consumers.

4. Click Create to create the distributed queue.

5. On the Thresholds & Quotas tab, define the following attributes for all of the distributed queue members:

   - Thresholds and quotas for messages and bytes (maximum number and high/low thresholds).
   - Whether or not bytes paging and/or messages paging is enabled on the distributed queue.

   If a distributed queue member's underlying physical queue already has a JMS Template with configured thresholds and quotas, these attributes will not apply to that queue member. For more information about these attributes, see "JMS Distributed Queue -> Configuration -> Thresholds & Quotas" in the Administration Console Online Help.

6. Click Apply to store new attribute values.

   Note: If you want to automatically create queue members on JMS servers that are part of a WebLogic Server cluster (for high availability) or on an single WebLogic Server instance that is not part of a cluster, see "Creating a Distributed Queue and Creating Members Automatically" on page 9-51.

7. Click the Configuration --> Members tab to define the queue members for the distributed queue.

8. Click the Configure a new Distributed Queue Member text link in the right pane. A Configuration dialog shows the tabs associated with configuring a new distributed queue member.

9. Define the attributes on the General tab according to the following table.

Table 9-15 Distributed Queue Member Attributes on the General Tab

Attribute	Description
Name	Uniquely identify the distributed queue member within a WebLogic Server domain.
JMSQueue	Select the underlying physical queue that is associated with the distributed queue member.

Table 9-15 Distributed Queue Member Attributes on the General Tab

Attribute	Description
Weight	Define the weight (that is, a measure of ability to handle message load) of the queue member with respect to other queue members in the distributed destination. The random distribution load-balancing algorithm uses the weight assigned to the physical destinations to compute a weighted distribution for the set of physical destinations. For more information, see "Developing a WebLogic JMS Application" in *Programming WebLogic JMS*.

10. Click Create to create the new distributed queue member. The new member is added to the Distributed Queue table.
11. Repeat steps 8–10 to continue adding members to the distributed queue.

Creating a JMS Distributed Queue Member

Follow these steps to add an existing physical queue as a member of distributed queue.

1. Expand the JMS →Distributed Destinations node. The Distributed Destinations table displays in the right pane showing all the distributed queues and topics.
2. Click the distributed queue that you want to add a member to. The Distributed Queue table shows all the distributed queue members that belong to the distributed queue.
3. Click the Configure a new Distributed Queue Member text link. A dialog shows the Configuration tab for configuring a new distributed queue member.
4. Define the configuration attributes for the distributed queue.
   - Uniquely identify the distributed queue member within a WebLogic Server domain.
   - Select the underlying physical queue that is associated with the distributed queue member.
   - Define the weight (that is, a measure of ability to handle message load) of the queue member with respect to other topic members in the distributed

destination. For more information about load balancing for distributed destinations, see "Developing a WebLogic JMS Application" in *Programming WebLogic JMS*.

For more information about distributed queue member attributes, see "JMS Distributed Queue Member -> Configuration" in the *Administration Console Online Help*.

5. Click Create to create a distributed queue member with the name you specified in the Name field. The new member is added to the Distributed Queue Member table in the right pane.
6. Click Apply to save any changes you made.

Deleting a JMS Distributed Queue Member

Follow these steps to delete a member of distributed queue configuration, and if necessary, to delete the queue member's underlying physical queue. For example, you must first delete a queue member from a distributed destination when planning to untarget or delete the JMS server that is hosting the member; otherwise, simply untargeting or deleting the JMS server can cause the configuration file (`config.xml`) to become unbootable.

Note: If you need to delete the entire distributed queue, then follow the directions for "Deleting a Distributed Destination" on page 9-59.

1. Expand the JMS →Distributed Destinations node. The Distributed Destinations table displays in the right pane showing all the distributed queues and topics.
2. Click the distributed queue that you want to delete a member from. The Distributed Queue table shows all the distributed queue members that belong to the distributed queue.
3. Click the Delete icon in the row of the distributed queue member that you want to delete. A dialog prompts you to confirm your deletion request.
4. If you want to also remove the underlying physical queue, select the Also Delete check box.
5. Click Remove to delete the distributed queue member (and the underlying physical queue if selected).

6. The Distributed Queue table redisplays in the right pane. The distributed queue member is deleted from the Distributed Queue table.

Creating a JMS Distributed Topic Member

Follow these steps to add an existing physical queue as a member of distributed queue.

1. Expand the JMS →Distributed Destinations node. The Distributed Destinations table displays in the right pane showing all the distributed queues and topics.
2. Click the distributed topic that you want to add a member to. The Distributed Topic table shows all the distributed topic members that belong to the distributed topic.
3. Click the Configure a new Distributed Topic Member text link. A dialog shows the Configuration tab for configuring a new distributed topic member.
4. Define the general configuration attributes for the distributed topic.
   - Uniquely identify the distributed topic member within a WebLogic Server domain.
   - Select the underlying physical topic that is associated with the distributed topic member.
   - Define the weight (that is, a measure of ability to handle message load) of the topic member with respect to other topic members in the distributed destination. For more information about load balancing distributed destinations, see "Developing a WebLogic JMS Application" in *Programming WebLogic JMS*.

   For more information about distributed topic member attributes, see "JMS Distributed Topic Member -> Configuration" in the *Administration Console Online Help*.
5. Click Create to create a distributed topic member with the name you specified in the Name field. The new member is added to the Distributed Topic table in the right pane.
6. Click Apply to save any changes you made.

Deleting a JMS Distributed Topic Member

Follow these steps to delete a member of distributed topic, and if necessary, to delete the topic member's underlying physical topic. For example, you must first delete a topic member from a distributed destination when planning to untarget or delete the JMS server that is hosting the member; otherwise, simply untargeting or deleting the JMS server can cause the configuration file (`config.xml`) to become unbootable.

Note: If you need to delete the entire distributed topic, then follow the directions for "Deleting a Distributed Destination" on page 9-59.

1. Expand the JMS node.
2. Expand the Distributed Destinations node. The Distributed Destinations table displays in the right pane showing all the distributed queues and topics.
3. Click the distributed topic that you want to delete a member from. The Distributed Topic table shows all the distributed topic members that belong to the distributed topic.
4. Click the Delete icon in the row of the distributed topic member that you want to delete. A dialog prompts you to confirm your deletion request.
5. If you want to also remove the underlying physical queue, select the Also Delete check box.
6. Click Remove to delete the distributed topic member (and the underlying physical topic if selected).
7. The Distributed Topic table redisplays in the right pane. The distributed topic member is deleted from the Distributed Topic table.

Deleting a Distributed Destination

If you need to delete an entire distributed destination, you must remove it in the following sequence:

1. Delete *all* the members of the distributed queue or distributed topic, as explained in the following sections:

- "Deleting a JMS Distributed Queue Member" on page 9-57
- "Deleting a JMS Distributed Topic Member" on page 9-59

2. Delete the distributed destination itself by expanding the JMS →Distributed Destinations node, and then clicking the trash icon next to the distributed destination that you want to delete.

   Note: You can only delete a distributed destination when *all* its members have been properly deleted.

3. You can delete the JMS template that is associated with the distributed destination. However, make sure that this template is not being used by other JMS servers or destinations. To delete a JMS Template, expand the JMS → Templates node, and then click the trash icon in the row of the JMS template that you want to delete.

Monitoring Distributed Destinations

When monitoring distributed destinations, you may see proxy topic members or system subscriptions, which are automatically created for the topic or queue members. For more information see, "Monitoring Distributed Destination System Subscriptions and Proxy Topic Members" on page 9-22.

Recovering from a WebLogic Server Failure

The following sections describe how to terminate a JMS application gracefully if a server fails and how to migrate JMS data after server failure.

Programming Considerations

You may want to program your JMS application to terminate gracefully in the event of a WebLogic Server failure. For example:

If a WebLogic Server Instance Fails and...	Then...
You are connected to the failed WebLogic Server instance	A `JMSException` is delivered to the connection exception listener. You must restart the application once the server is restarted or replaced.
You are not connected to the failed WebLogic Server instance	You must re-establish everything once the server is restarted or replaced.
A JMS Server is targeted on the failed WebLogic Server instance	A `ConsumerClosedException` is delivered to the session exception listener. You must re-establish any message consumers that have been lost.

Migrating JMS Data to a New Server

WebLogic JMS uses the migration framework implemented in the WebLogic Server core, which allows WebLogic JMS to properly respond to migration requests and bring a WebLogic JMS server online and offline in an orderly fashion. This includes both scheduled migrations as well as migrations in response to a WebLogic Server failure.

Once properly configured, a JMS server and all of its destination members can migrate to another WebLogic Server within a cluster.

You can recover JMS data from a failed WebLogic Server by starting a new server and doing one or more of the tasks in the following table:

If your JMS application uses. . .	Perform the following task. . .
Persistent messaging—JDBC Store	■ If the JDBC database store physically exists on the failed server, migrate the database to a new server and ensure that the JDBC connection pool URL attribute reflects the appropriate location reference. ■ If the JDBC database does not physically exist on the failed server, access to the database has not been impacted, and no changes are required.
Persistent messaging—File Store	Migrate the file to the new server, ensuring that the pathname within the WebLogic Server home directory is the same as it was on the original server.

If your JMS application uses. . .	Perform the following task. . .
Transactions	Migrate the transaction log to the new server by copying all files named `<servername>*.tlog`. This can be accomplished by storing the transaction log files on a dual-ported disk that can be mounted on either machine, or by manually copying the files. If the files are located in a different directory on the new server, update that server's `TransactionLogFilePrefix` server configuration attribute before starting the new server. **Note:** If migrating following a system crash, it is very important that the transaction log files be available when the server is restarted at its new location. Otherwise, transactions in the process of being committed at the time of the crash might not be resolved correctly, resulting in data inconsistencies. All uncommitted transactions are rolled back.

Note: JMS persistent stores can increase the amount of memory required during initialization of WebLogic Server as the number of stored messages increases. When rebooting WebLogic Server, if initialization fails due to insufficient memory, increase the heap size of the Java Virtual Machine (JVM) proportionally to the number of messages that are currently stored in the JMS persistent store and try the reboot again.

For information about starting a new WebLogic Server, refer to "Starting and Stopping WebLogic Servers" on page 2-1. For information about recovering a failed server, refer to Recovering Failed Servers in the *Configuring and Managing WebLogic Domains* guide.

For more information about migratable targets, see "Configuring WebLogic Migratable Services" in *Using WebLogic Server Clusters*.

CHAPTER

10 Using the WebLogic Messaging Bridge

The following sections explain how to configure and manage a WebLogic Messaging Bridge:

What Is a Messaging Bridge?

The WebLogic Messaging Bridge allows you to configure a forwarding mechanism between any two messaging products—thereby, providing interoperability between separate implementations of WebLogic JMS or between WebLogic JMS and another messaging product. You can use the WebLogic Messaging Bridge to integrate your messaging applications between:

- Any two implementations of WebLogic JMS, including those from separate releases of WebLogic Server.

- WebLogic JMS implementations that reside in separate WebLogic domains.
- WebLogic JMS with a third-party JMS product (for example, MQSeries).
- WebLogic JMS with non-JMS messaging products (only by using specialized adapters that are not provided with WebLogic Server).

A messaging bridge consists of two destinations that are being bridged: a source destination *from which* messages are received, and a target destination *to which* messages are forwarded. For WebLogic JMS and third-party JMS products, a messaging bridge communicates with source and target destinations using the resource adapters provided with WebLogic Server. For non-JMS messaging products, a custom adapter must be obtained from a third-party OEM vendor or by contacting BEA Professional Services in order to access non-JMS source or target destinations.

Source and target bridge destinations can be either queues or topics. You can also specify a quality of service (QOS), as well as message filters, transaction semantics, and connection retry policies. Once a messaging bridge is configured, it is easily managed from the Administration Console, including temporarily suspending bridge traffic whenever necessary, tuning the execute thread pool size to suit your implementation, and monitoring the status of all your configured bridges.

Messaging Bridge Configuration Tasks

Before you can deploy a messaging bridge, you need to configure its required components:

- "About the Bridge's Resource Adapters" on page 10-3
- "Deploying the Bridge's Resource Adapters" on page 10-4
- "Configuring the Source and Target Bridge Destinations" on page 10-5
- "Configuring a Messaging Bridge Instance" on page 10-11

About the Bridge's Resource Adapters

The messaging bridge uses resource adapters to communicate with source and target JMS destinations. You need to associate both the source and target JMS destinations with a *supported* adapter in order for the bridge to communicate with them. The JNDI name for the adapter is configured as part of the adapter's deployment descriptor.

Note: Although WebLogic JMS includes a "General Bridge Destination" framework for accessing non-JMS messaging products, WebLogic Server does not provide supported adapters for such products. Therefore, you must obtain a custom adapter from a third-party OEM vendor and consult their documentation for configuration instructions. You can also contact BEA Professional Services for information about obtaining a custom adapter.

The supported adapters are located in the *`WL_HOME\server\lib`* directory and are described in the following table.

Table 10-1 Messaging Bridge Adapters and JNDI Names

Adapter	JNDI Name	Description
`jms-xa-adp.rar`	`eis.jms.WLSConnectionFactoryJNDIXA`	Provides transaction semantics via the `XAResource`. Used when the required QOS is *Exactly-once*. This envelops a received message and sends it within a user transaction (XA/JTA). The following requirements are necessary in order to use this adapter: ■ Any WebLogic Server implementation being bridged must be release 6.1 or later. ■ The source and target JMS connection factories must be configured to use the `XAConnectionFactory`. **Note:** Before deploying this adapter, refer to "Using the Messaging Bridge to Interoperate with Different WebLogic Server Releases and Domains" on page 10-17 for specific transactional configuration requirements and guidelines.

Table 10-1 Messaging Bridge Adapters and JNDI Names

Adapter	JNDI Name	Description
`jms-notran-adp.rar`	`eis.jms.WLSConnectionFactoryJNDINoTX`	Provides no transaction semantics. Used when the required QOS is *Atmost-once or Duplicate-okay*. If the requested QOS is *Atmost-once*, the adapter uses the `AUTO_ACKNOWLEDGE` mode. If the requested QOS is *Duplicate-okay*, `CLIENT_ACKNOWLEDGE` is used. **Note:** For more information about the acknowledge modes used in non-transacted sessions, see "WebLogic JMS Fundamentals" in *Programming WebLogic JMS*.
`jms-notran-adp51.rar`	`eis.jms.WLS51ConnectionFactoryJNDINoTX`	Provides interoperability when either the source or target destination is WebLogic Server 5.1. This adapter provides no transaction semantics; therefore, it only supports a QOS of *Atmost-once* or *Duplicate-okay*. If the requested QOS is *Atmost-once*, the adapter uses the `AUTO_ACKNOWLEDGE` mode. If the requested QOS is *Duplicate-okay*, `CLIENT_ACKNOWLEDGE` is used. Deprecated, see "WebLogic Server 5.1 End-of-Life Announcement" in *Supported Configurations*.

You will specify the appropriate adapter by its JNDI name when you configure each source and target bridge destination on the Administration Console.

Deploying the Bridge's Resource Adapters

Before you configure the messaging bridge components, deploy the appropriate resource adapters in the WebLogic Server domain that is hosting the messaging bridge, using either of the following methods:

- On the Administration Console — Select the Domain in which you will be deploying the adapters in, and then select Deployments →Applications option, and then select the appropriate `RAR` adapter file, as defined in Table 10-1, "Messaging Bridge Adapters and JNDI Names," on page 10-3.
  - `jms-xa-adp.rar`
  - `jms-notran-adp.rar`
  - `jms-notran-adp51.rar (Deprecated)`
- Using the Auto Deployment feature — This method is used for quickly deploying an application on the administration server. By copying the adapters to the local `\applications` directory of the administration server, they will be automatically deployed if the server is already running. Otherwise, they will be deployed the next time you start WebLogic Server. The auto deployment method is used only in a single-server development environment for testing an application, and is not recommended for use in production mode.

Note: When configuring a messaging bridge to interoperate between WebLogic Server release 7.0 and release 5.1, then the release 5.1 resource adapter (`jms-notran-adp51.rar`) and the non-transaction adapter (`jms-notran-adp.rar`) must be deployed on the 7.0 domain running the messaging bridge.

For step-by-step instructions on deployment tasks using the Administration Console, or for more information on using the auto-deployment feature, see "WebLogic Server Deployment" in *Developing WebLogic Server Applications*.

Configuring the Source and Target Bridge Destinations

A messaging bridge connects two actual destinations that are mapped to bridge destinations: a source destination *from which* messages are received, and a target destination *to which* messages are sent. Depending on the messaging products that need to be bridged, there are two types of bridge destinations:

- JMS Bridge Destination – For JMS messaging products, whether it is a WebLogic JMS implementation or a third-party JMS provider, you need to configure a `JMSBridgeDestination` instance for each actual source and target JMS destination being mapped to a messaging bridge.

- General Bridge Destination – For non-JMS messaging products, you need to configure a generic `BridgeDestination` instance for each actual source and target destination being mapped to a messaging bridge.

Before starting the procedure in this section, refer to the "Using the Messaging Bridge to Interoperate with Different WebLogic Server Releases and Domains" on page 10-17 for specific configuration requirements and guidelines.

Configuring a JMS Bridge Destination

A `JMSBridgeDestination` instance defines a unique name for the actual JMS queue or topic destination within a WebLogic domain, the name of the adapter used to communicate with the specified destination, property information to pass to the adapter (Connection URL, Connection Factory JNDI Name, etc.), and, optionally, a user name and password.

You need to configure a `JMSBridgeDestination` instance for each actual source and target JMS destination to be mapped to a messaging bridge. Therefore, when you finish defining attributes for a source JMS bridge destination, repeat these steps to configure a target JMS bridge destination, or vice versa. You will designate the source and target JMS Bridge Destinations in "Configuring a Messaging Bridge Instance" on page 10-11.

To configure a JMS bridge destination, follow these steps.

1. In the Administration Console, click the Messaging Bridge node.
2. Click the JMS Bridge Destinations node to open the Bridge Destinations tab in the right pane.
3. In the right pane, click the Configure a new JMS Bridge Destination link. A Configuration dialog shows the tabs associated with configuring a new JMS bridge destination.
4. Define the attributes in the Configuration tab.

   The following table describes the attributes you set on the Configuration tab.

Table 10-2 JMS Bridge Destination Attributes on the Configuration Tab

Attribute	Description
Name	A JMS bridge destination name for the actual JMS destination being mapped to the bridge. This name must be unique across a WebLogic domain. For example, if you are bridging between WebLogic Server releases 6.1 and 7.0, for the source destination you could change the default bridge destination name to "70to61SourceDestination". Then, when you create the corresponding target destination, you could name it as "70to61TargetDestination". Once the bridge destinations are configured, these names are listed as options in the Source Destination and Target Destination attributes on the Bridges → General tab.
Adapter JNDI Name	The JNDI name of the adapter used to communicate with the bridge destinations. For more information on which adapter name to enter, see "Messaging Bridge Adapters and JNDI Names" on page 10-3.
Adapter Classpath	When connecting to a destination that is running on WebLogic Server 6.0 or earlier, the bridge destination must supply a `CLASSPATH` that indicates the locations of the classes for the earlier WebLogic Server implementation. When connecting to a third-party JMS provider, the bridge destination must supply the provider's `CLASSPATH` in the WebLogic Server `CLASSPATH`.
Connection URL	The URL of the JNDI provider used to look up the connection factory and destination.
Initial Context Factory	The factory used to get the JNDI context.
Connection Factory JNDI Name	The JMS connection factory used to create a connection for the actual JMS destination being mapped to the JMS bridge destination. **Note:** In order to use the *Exactly-once* QOS, the connection factory has to be a XAConnection Factory. For more information about connection factory and QOS requirements, see "Messaging Bridge Attributes on the General Tab" on page 10-11.

Table 10-2 JMS Bridge Destination Attributes on the Configuration Tab

Attribute	Description
Destination JNDI Name	The JNDI name of the actual JMS destination being mapped to the JMS bridge destination.
Destination Type	Select either a Queue or Topic destination type.
User Name and Password	The user name and password that the messaging bridge will give to the bridge adapter. **Note:** All operations done to the specified destination are done using that user name and password. Therefore, the User Name/Password for the source and target destinations must have permission to access the underlying JMS destinations in order for the messaging bridge to work.

5. Click Create to create the JMS bridge destination.
6. When you finish defining attributes for a source JMS bridge destination, repeat these steps to configure a target JMS bridge destination, or vice versa.

Configuring a General Bridge Destination

A general `BridgeDestination` instance defines a unique name for the actual queue or topic destination within the WebLogic domain, the name of the resource adapter used to communicate with the specified destination, a list of properties to pass to the adapter, and, optionally, a user name and password.

Note: Although WebLogic JMS includes a "General Bridge Destination" framework for accessing non-JMS messaging products, WebLogic Server does not provide supported adapters for such products. Therefore, you must obtain a custom adapter from a third-party OEM vendor and consult their documentation for configuration instructions. You can also contact BEA Professional Services for information about obtaining a custom adapter.

You need to configure a `BridgeDestination` instance for each actual source and target destination to be mapped to a messaging bridge. You will designate the source and target general bridge destinations in "Configuring a Messaging Bridge Instance" on page 10-11.

To configure a general bridge destination, follow these steps:

1. In the Administration Console, click the Messaging Bridge node.
2. Click the General Bridge Destinations node to open the Bridge Destinations tab in the right pane.
3. In the right pane, click the Configure a new General Bridge Destination link. A Configuration dialog shows the tabs associated with configuring a new general bridge destination.
4. Define the attributes in the Configuration tab.

   The following table describes the attributes you set on the Configuration tab.

Table 10-3 General Bridge Destination Attributes on the Configuration Tab

Attribute	**Description**
Name	A bridge destination name for the actual destination being mapped to the bridge. This name must be unique across a WebLogic domain. For example, if you are bridging between WebLogic Server releases 6.1 and 7.0, for the source destination you could change the default bridge destination name to "70to61SourceDestination". Then, when you create the corresponding target destination, you could name it as "70to61TargetDestination". Once the bridge destinations are configured, these names are listed as options in the Source Destination and Target Destination attributes on the Bridges → General tab.
Adapter JNDI Name	A bridge destination must supply the JNDI name of the adapter used to communicate with the bridge destinations. WebLogic Server does not provide adapters for non-JMS messaging products. Therefore, you must use a specialized adapter from a third-party OEM vendor, or contact BEA Professional Services to obtain a custom adapter.
Adapter Classpath	Defines the `CLASSPATH` of the bridge destination. This is attribute is mainly used to connect to a destination running on WebLogic Server 6.0 or earlier. When connecting to a third-party product, you must supply the product's `CLASSPATH` in the WebLogic Server `CLASSPATH`.

Table 10-3 General Bridge Destination Attributes on the Configuration Tab

Attribute	Description
Properties	Specifies all the properties defined for a bridge destination. Each property must be separated by a semicolon (for example, `DestinationJNDIName=myTopic;DestinationType =topic;`). For non-JMS messaging products that use adapters provided by a third-party OEM vendor, you should consult the vendor's documentation for property configuration instructions. The following properties are required for all JMS implementations: `ConnectionURL`= URL used to establish a connection to the destination. `InitialContextFactory=` Factory used to get the JNDI context. `ConnectionFactoryJNDIName`= The JMS connection factory used to create a connection for the actual JMS destination being mapped to the JMS bridge destination. `DestinationJNDIName`= The JNDI name of the actual JMS destination being mapped to the JMS bridge destination. `DestinationType`= Queue or topic.
User Name and Password	The user name that the messaging bridge will give to the bridge adapter. **Note:** All operations done to the specified destination are done using this user name and password. Therefore, the User Name/Password for the source and target bridge destinations must have permission to access the underlying source and target destinations in order for the messaging bridge to work.

5. Click Create to create the general bridge destination.
6. When you finish defining attributes for a source general bridge destination, repeat these steps to configure a target general bridge destination, or vice versa.

Configuring a Messaging Bridge Instance

A messaging bridge instance communicates with the configured source and target bridge destinations. For each mapping of a source destination to a target destination, whether it is another WebLogic JMS implementation, a third-party JMS provider, or another non-JMS messaging product, you must configure a `MessagingBridge` instance via the Administration Console. Each `MessagingBridge` instance defines the source and target destination for the mapping, a message filtering selector, a QOS, transaction semantics, and various reconnection parameters.

Before starting the procedure in this section, refer to the "Using the Messaging Bridge to Interoperate with Different WebLogic Server Releases and Domains" on page 10-17 or "Using the Messaging Bridge to Access a Third-Party Messaging Provider" on page 10-22 for specific configuration requirements and guidelines.

To configure a messaging bridge, follow these steps:

1. In the Administration Console, click the Messaging Bridge node.
2. Click the Bridges node to open the Bridges tab in the right pane.
3. Click the Configure a new Messaging Bridge link in the right pane. A Configuration dialog shows the tabs associated with configuring a new messaging bridge.
4. Define the attributes in the General tab.

   The following table describes the attributes you set on the General tab.

Table 10-4 Messaging Bridge Attributes on the General Tab

Attribute	Description
Name	Enter a name for the messaging bridge that is unique across a WebLogic domain.
Source Destination	Select the source destination *from which* messages are received by the messaging bridge. For example, for a JMS messaging bridge, you should select the "JMS Source Bridge Destination" name that you created on the JMS Bridge Destination → Configuration tab.

Table 10-4 Messaging Bridge Attributes on the General Tab

Attribute	Description
Target Destination	Select the target destination *to which* messages are sent from the messaging bridge. For example, for a JMS messaging bridge, you should select the "JMS Target Bridge Destination" name that you created on the JMS Bridge Destination → Configuration tab.
Selector	Allows you to filter the messages that are sent across the messaging bridge. Only messages that match the selection criteria are sent across the messaging bridge. For queues, messages that do not match the selection criteria are left behind and accumulate in the queue. For topics, messages that do not match the connection criteria are dropped. For more information on using selectors to filter messages, see "Developing a WebLogic JMS Application" in *Programming WebLogic JMS*.
Quality Of Service (QOS)	Select a QOS guarantee for forwarding a message across a messaging bridge. The valid qualities of service are: *Exactly-once*—Each message will be sent exactly once. This is the highest quality of service. In order to use this QOS: ■ Any WebLogic Server implementation must be release 6.1 or later. ■ The source and target JMS connection factories must be configured to use the `XAConnectionFactory`. ■ The transaction `jms-xa-adp.rar` adapter must be deployed and identified in the Adapter JNDI Name attribute as "`eis.jms.WLSConnectionFactoryJNDIXA`" for both the source and target destinations. *Atmost-once*—Each message is sent at most one time. Some messages may not be delivered to the target destination. *Duplicate-okay*—Each message is sent at least one time. Duplicate messages can be delivered to the target destination.

Table 10-4 Messaging Bridge Attributes on the General Tab

Attribute	Description
QOS Degradation Allowed	When selected, the messaging bridge automatically degrades the requested QOS when the configured one is not available. If this occurs, a message is delivered to the WebLogic startup window (or log file). If this option is not selected (false), and the messaging bridge cannot satisfy the requested QOS, it will result in an error and the messaging bridge will not start.
Maximum Idle Time (seconds)	For bridges running in asynchronous mode, this is the maximum amount of time, in seconds, the messaging bridge sits idle before checking the health of its connections. For bridges running in synchronous mode, this dictates the amount of time the messaging bridge can block on a receive call if no transaction is involved.
Asynchronous Mode Enabled	Defines whether a messaging bridge works in asynchronous mode. Messaging bridges that work in asynchronous mode (true) are driven by the source destination. The messaging bridge listens for messages and forwards them as they arrive. When the value is false, the bridge works in synchronous mode even if the source supports asynchronous receiving. **Note:** For a messaging bridge with a QOS of *Exactly-once* to work in asynchronous mode, the source destination has to support the `MDBTransaction` interface described in the `weblogic.jms.extensions` Javadoc. Otherwise, the bridge automatically switches to synchronous mode if it detects that `MDBTransactions` are not supported by the source destination. For more information about `MDBTransactions`, see "Using Message-Driven Beans" in *Programming WebLogic Enterprise Java Beans*.

Table 10-4 Messaging Bridge Attributes on the General Tab

Attribute	Description
Durability Enabled	This attribute is used only for JMS topics or for destinations with similar characteristics as a JMS topic. By enabling durability, a messaging bridge creates a durable subscription for the source destination. This allows the source JMS implementation to save messages that are sent to it when the bridge is not running. The bridge will then forward these messages to the target destination once it is restarted. If this attribute is not selected, messages that are sent to the source JMS topic while the bridge is down cannot be forwarded to the target destination. **Note:** If a bridge must be taken permanently offline, you must delete any durable subscriptions that use the bridge. For information on deleting durable subscribers, see "Deleting Durable Subscriptions" in *Programming WebLogic JMS*.
Started	Indicates the initial state of the messaging bridge when it is configured and whenever the server is restarted. You can also use this field to dynamically start and stop the messaging bridge. To stop the bridge, clear the check box. Conversely, reselect the check box to restart the bridge. **Note:** Unless there is a configuration issue that prevents the messaging bridge from starting, this field indicates the expected run-time state of the messaging bridge. For information on monitoring all the configured messaging bridges in your domain, see "Monitoring Messaging Bridges" on page 10-24.

5. Click Create to create the messaging bridge.
6. On the Connection Retry tab, define the bridge's reconnection time intervals according to the following table.

   The source and target destinations for a messaging bridge will not always be available. As such, the messaging bridge must be able to reconnect to the destination at some periodic interval. These attributes govern the time between reconnection attempts.

Table 10-5 Messaging Bridge Attributes on the Connection Retry Tab

Attribute	Description
Minimum Delay (seconds)	The minimum delay, in seconds, between reconnection attempts. When a messaging bridge boots and cannot connect to a destination, or a connection is lost and the messaging bridge is first attempting to reconnect, it attempts to reconnect in this specified amount of seconds.
Incremental Delay (seconds)	The delay increment, in seconds, between reconnection attempts. Each time a bridge fails to reconnect, it adds this amount of seconds to the delay before making its next reconnection attempt.
Maximum Delay (seconds)	The maximum delay, in seconds, between reconnection attempts. Each reconnection attempt is delayed further by the Incremental Delay amount of seconds, but it is never delayed by more than this value.

7. Click Apply to store new attribute values.

8. On the Transactions tab, define the transaction attributes for the messaging bridge according to the following table.

Table 10-6 Messaging Bridge Attributes on the Transactions Tab

Attribute	Description
Transaction Timeout	Defines the number of seconds the transaction manager waits for each transaction before timing it out. Transaction timeouts are used when a bridge's quality of service requires two-phase transactions.
Batch Size	Defines the number of messages that the messaging bridge transfers within one transaction. Batch Size only applies to bridges that work in synchronous mode and whose quality of service require two-phase transactions.

Table 10-6 Messaging Bridge Attributes on the Transactions Tab

Attribute	Description
Batch Interval (milliseconds)	Defines the maximum time, in milliseconds, that the bridge waits before sending a batch of messages in one transaction, regardless of whether the Batch Size amount has been reached or not. The default value of -1 indicates that the bridge will wait until the number of messages reaches the *Batch Size* before it completes a transaction. Batch Interval only applies to bridges that work in synchronous mode and whose quality of service require two-phase transactions.

9. Click Apply to store new attribute values.

10. On the Targets tab, assign WebLogic Server instances to associate with the messaging bridge according to the following table.

Table 10-7 Messaging Bridge Attributes on the Targets Tab

Attribute	Description
Migratable Targets	Defines a WebLogic Server migratable target where the messaging bridge will be deployed. When WebLogic Server is first booted, the messaging bridge initially is available only on the *user-preferred* server. Afterwards, the bridge can be migrated to another server in the migratable target using either the Administration Console or the command-line tool. For more information, see "Migration for Pinned Services" in *Using WebLogic Server Clusters*.
Clusters	Defines a WebLogic Server cluster where the messaging bridge will be deployed. The messaging bridge will be available on all servers in the selected cluster.
Servers	Defines the WebLogic Servers where the messaging bridge will be deployed. The messaging bridge will be available on all the selected WebLogic Servers.

11. Click Apply to store new attribute values.

Using the Messaging Bridge to Interoperate with Different WebLogic Server Releases and Domains

The following interoperability guidelines apply when using the messaging bridge to access JMS destinations in different releases of WebLogic Server and in other WebLogic Server domains.

- "Naming Guidelines for WebLogic Servers and Domains" on page 10-17
- "Message Properties" on page 10-18
- "Configuring Interoperability for WebLogic Domains" on page 10-18
- "Using the Messaging Bridge To Access Destinations In a Release 6.1 or Later Domain" on page 10-19
- "Using the Messaging Bridging To Access Destinations In a Release 6.0 Domain" on page 10-20
- "Using the Messaging Bridging To Access Destinations In a Release 5.1 Domain" on page 10-21

Note: When the messaging bridge is used to communicate between two domains running different releases of Weblogic Server, a best-practice recommendation is for the messaging bridge to be configured to run on the domain using the latest release of Weblogic Server.

Naming Guidelines for WebLogic Servers and Domains

Unique naming rules apply to all WebLogic Server deployments if more than one domain is involved. Therefore, make sure that:

- WebLogic Server instances and domain names are unique.
- WebLogic JMS server names are unique name across domains.

- All JMS connection factories targeted to servers in a cluster are uniquely named.
- If a JMS file store is being used for persistent messages, the JMS file store name must be unique across domains.

Message Properties

Message properties are inherited from the `Default Delivery Mode` attribute on the connection factory used when a message is forwarded to its target destination. If the `Default Delivery Mode` is `Persistent`, a non-persistent message is forwarded as a persistent message resulting in a significant performance loss.

If you configure a bridge instance to forward non-persistent messages, configure and use a connection factory that has the `Default Delivery Mode` set to `Non-Persistent`.

Configuring Interoperability for WebLogic Domains

Unless the Exactly-once QOS (quality of service) is required for handling two-phase transactions sent across the messaging bridge, there are no special security configuration requirements for the bridge to interoperate between two release 6.1 or later domains.

However, you *must* follow these steps when a bridge running on release 7.0 domain must handle transactional messages (using the Exactly-once QOS) between two release 6.1 or later domains

1. For each server participating in the transaction, set the `Security Interoperability Mode` flag according to Table 7-2, "Security Interoperability Settings for Inter-Domain Transactions," on page 7-9 before rebooting. See "Using Security Interoperability Mode" on page 7-8.

   Note: When `Security Interoperability Mode` is set to performance, you are not required to set domain trust between the domains.

2. Configure domain trust for the all participating bridge domains.

   a. In all participating WebLogic Server 6.x domains, change the password for the `system` user to the same value in all participating domains on the Security→Users tab in the Administration Console. See Changing the System

Password at `http://e-docs.bea.com/wls/docs61/adminguide/cnfgsec.html#cnfgsec003`.

b. Establish domain trust by setting a security credential for all domains to the same value in all participating domains. If you have participating 6.x domains, set the security credential for all domains to the same value as the `system` password in all participating WebLogic Server 6.x domains.

- For 7.x domains, see Enabling Trust Between WebLogic Domains in *Managing WebLogic Security.*
- For 8.x domains, see Enabling Trust Between WebLogic Domains in *Managing WebLogic Security.*
- For 9.x domains, see Enable trust between domains in *Administration Console Online Help.*

Using the Messaging Bridge To Access Destinations In a Release 6.1 or Later Domain

Use these guidelines when configuring a messaging bridge on a release 7.0 domain to provide “Exactly-once” transactional message communication between two release 6.1 or later domains.

Note: The *Exactly-once* quality of service for two-phase transactions is only supported for release 6.1 or later.

- If a JMS file store is being used for persistent messages, the JMS file store name must be unique across WebLogic domains, as described in “Naming Guidelines for WebLogic Servers and Domains” on page 10-17.
- Make sure that interoperability between the domains is correctly configured, as described in “Configuring Interoperability for WebLogic Domains” on page 10-18.
- Make sure that the transaction connection factory is enabled for both domains by selecting the XAConnection Factory Enabled check box on the Services →JMS →Connection Factories →Configuration →Transactions tab.

- Deploy the transaction resource adapter, `jms-xa-adp.rar`, on the 7.0 bridge domain, as described in "Deploying the Bridge's Resource Adapters" on page 10-4.
- When configuring the JMS bridge destinations, as described in "Configuring a JMS Bridge Destination" on page 10-6, do the following for both the source and target destinations:
  - In the Adapter JNDI Name field, identify the transaction adapter's JNDI name, `eis.jms.WLSConnectionFactoryJNDIXA`.
  - Do not enter anything in the Adapter Classpath field.
- On the Messaging Bridge →Configuration →General tab, select a Quality Of Service of *Exactly-once*, as described in "Configuring a Messaging Bridge Instance" on page 10-11.

Using the Messaging Bridging To Access Destinations In a Release 6.0 Domain

When configuring a messaging bridge involves interoperability between WebLogic Server 7.0 and a release 6.0 domain, you must configure the following items on the release 7.0 domain that the bridge is running on:

Note: The *Exactly-once* QOS for transactions is not supported for WebLogic Server 6.0. For more information on the bridge QOS options, see "Messaging Bridge Attributes on the General Tab" on page 10-11.

- Deploy the non-transaction resource adapter, `jms-notran-adp.rar` on the 7.0 bridge domain, as described in "Deploying the Bridge's Resource Adapters" on page 10-4.
- When configuring the JMS source and target destinations, as described in "Configuring a JMS Bridge Destination" on page 10-6, do the following:

  In the Adapter JNDI Name field:

  - For the source and target destinations, specify the non-transaction adapter's JNDI name as `eis.jms.WLSConnectionFactoryJNDINoTX`.

  In the Adapter Classpath field:

  - For the 7.0 destination, leave the field blank.

- For the 6.0 destination, indicate the location of the classes for the WebLogic Server 6.0 release.

  For example, if you have WebLogic Server 6.0 GA installed in a directory named `WL60_HOME`, then set the Adapter Classpath as follows for the 6.0 JMS bridge destination:

  ```
  WL60_HOME\lib\weblogic60.jar
  ```

- On the Messaging Bridge →Configuration →General tab, select a Quality Of Service of *Atmost-once* or *Duplicate-okay*, as described in "Configuring a Messaging Bridge Instance" on page 10-11.

Using the Messaging Bridging To Access Destinations In a Release 5.1 Domain

Note: The `jms51-interop.jar` file and `jms-notran-adp51.rar` file are deprecated, see "WebLogic Server 5.1 End-of-Life Announcement" in *Supported Configurations.*

When configuring a messaging bridge involves interoperability between WebLogic Server 7.0 and release 5.1, you must configure the following on the release 7.0 domain that the bridge is running on:

Note: The *Exactly-once* QOS for transactions is not supported for WebLogic Server 5.1. For more information on the bridge QOS options, see "Messaging Bridge Attributes on the General Tab" on page 10-11.

- The `jms51-interop.jar` file in the `WL_HOME\lib` directory must be in the `CLASSPATH` of the WebLogic Server 7.0 implementation.
- The release 5.1 resource adapter (`jms-notran-adp51.rar`) and the non-transaction adapter (`jms-notran-adp.rar`) must be deployed on the 7.0 bridge domain, as described in "Deploying the Bridge's Resource Adapters" on page 10-4.
- When configuring the JMS source and target destinations, as described in "Configuring a JMS Bridge Destination" on page 10-6, do the following:

  In the Adapter JNDI Name field:

- For the 7.0 destination, specify the non-transaction adapter's JNDI name as `eis.jms.WLSConnectionFactoryJNDINoTX`.
- For the 5.1 destination, specify the 5.1 adapter's JNDI name as `eis.jms.WLS51ConnectionFactoryJNDINoTX`.

In the Adapter Classpath field:

- For the 7.0 destination, leave the field blank.
- For the 5.1 destination, indicate the location of the classes for the WebLogic Server 5.1 release, as well as the location of the `jms51-interop.jar` file for the 7.0 release.

  For example, if you have WebLogic Server 5.1 GA installed in a directory named *`WL51_HOME`* and your WebLogic Server 7.0 release is installed in *`WL81_HOME`*, then set the Adapter Classpath as follows for the 5.1 destination:

  ```
  WL51_HOME\classes;WL51_HOME\lib\weblogicaux.jar;
  WL70_HOME\server\lib\jms51-interop.jar
  ```

  Note: If your implementation is using a 5.1 Service Pack, the corresponding *`sp.jar`* files must also be added to the Adapter Classpath field.

- On the Messaging Bridge →Configuration →General tab, select a Quality Of Service of *Atmost-once* or *Duplicate-okay*, as described in "Configuring a Messaging Bridge Instance" on page 10-11.

Using the Messaging Bridge to Access a Third-Party Messaging Provider

When configuring a messaging bridge involves interoperability with a third-party messaging provider, you must configure the following:

- Before starting WebLogic Server:
  - Supply the provider's `CLASSPATH` in the WebLogic Server `CLASSPATH`.
  - Include the `PATH` of any native code required by the provider's client-side libraries in the WebLogic Server system `PATH`. (This variable may vary depending on your operating system.)

- In the `JMSBridgeDestination` instance for the third-party messaging product being bridged, provide *vendor-specific* information in the following attributes:
  - Connection URL
  - Initial Context Factory
  - Connection Factory JNDI Name
  - Destination JNDI Name

Note: The messaging bridge cannot provide the "Exactly-once" quality of service when the source and target bridge destinations are located on the same resource manager (that is, when the bridge is forwarding a global transaction that is using the XA resource of the resource manager). For example, when using MQ Series, it is not possible to use the same Queue Manager for the source and target bridge destinations.

For more information on configuring the remaining attributes for a JMS Bridge Destination, see "Configuring a JMS Bridge Destination" on page 10-6.

Managing a Messaging Bridge

Once a messaging bridge is up and running, it can managed from the Administration Console.

- Stopping and Restarting a Messaging Bridge
- Monitoring Messaging Bridges
- Configuring the Execute Thread Pool Size

Stopping and Restarting a Messaging Bridge

To temporarily suspend and restart an active messaging bridge:

1. Expand the Messaging Bridge node and the Bridges node.
2. Select the messaging bridge instance that you want to stop.

3. On the Configuration General tab, clear the Started check box to stop the bridge.
4. To restart the bridge, select the Started check box.

Monitoring Messaging Bridges

You can monitor the status of all the messaging bridges in your domain from the Administration Console:

1. Expand the Servers node and select the server instance where the messaging bridges are configured.
2. A dialog displays in the right pane showing the tabs associated with the selected server instance.
3. Select the Services tab.
4. Select the Bridge tab.
5. Click the Monitoring all Messaging Bridge Runtimes text link to display the monitoring data.
6. A table displays showing all the messaging bridge instances for the server and their status (either as running or not running).

Configuring the Execute Thread Pool Size

You can configure the default execute thread pool size for your messaging bridges from the Administration Console. For example, you may want to increase the default size to reduce competition from the WebLogic Server default thread pool. Entering a value of -1 disables this thread pool and forces a messaging bridge to use the WebLogic Server default thread pool.

1. Expand the Servers node.
2. Select the server instance where the messaging bridge is configured. A dialog displays in the right pane showing the tabs associated with the selected server instance.
3. Select the Services tab.

4. Select the Bridge tab.
5. Enter a new value in the Messaging Bridge Thread Pool Size field.
6. Click Apply to save your changes.

For more information about tuning execute threads, see "Tuning WebLogic Server Applications" in the *Performance and Tuning Guide*.

CHAPTER

11 Managing JNDI

The following sections describe how to manage JNDI:

- "Overview of JNDI Management" on page 11-1
- "Viewing the JNDI Tree" on page 11-2
- "Loading Objects in the JNDI Tree" on page 11-2

Overview of JNDI Management

You use the Administration Console to manage JNDI. The JNDI API enables applications to look up objects—such as Data Sources, EJBs, JMS, and MailSessions—by name. The JNDI tree is represented by the left pane in the Administration Console.

For additional information, see Programming WebLogic JNDI at http://e-docs.bea.com/wls/docs70/jndi/index.html.

What Do JNDI and Naming Services Do?

JNDI provides a common-denominator interface to many existing naming services, such as LDAP (Lightweight Directory Access Protocol) and DNS (Domain Name System). These naming services maintain a set of bindings, which relate names to objects and provide the ability to look up objects by name. JNDI allows the components in distributed applications to locate each other.

Viewing the JNDI Tree

To view the objects in the WebLogic Server JNDI tree for a specific server, do the following:

1. Right-click the server node in the left pane. This displays a pop-up menu.
2. Select JNDI Tree. The JNDI tree for this server displays in the right pane.

Loading Objects in the JNDI Tree

Using the Administration Console, you load WebLogic Server J2EE services and components, such as RMI, JMS, EJBs, and JDBC Data Sources, in the JNDI tree.

To load an object in the JNDI tree, choose a name under which you want the object to appear in the JNDI tree. Then enter that name in the JNDI Name attribute field when you create the object. When the object is loaded, JNDI provides a path to the object.

To verify if an object has been loaded, see "Viewing the JNDI Tree".

For more information on configuring objects, see Table 11-1 Objects in JNDI Tree.

Table 11-1 Objects In JNDI Tree

Service	Bound Object w/Link to Online Help
EJB	EJB Deployment Descriptor Editor
JDBC DataSource	JDBC Data Source and JDBC Transaction (Tx) Data Source
JMS Connection Factory	JMS Connection Factories
Web Services	Web Application Deployment Descriptor Editor
Mail	MailSession
Deployment Descriptors	BEA WebLogic J2EE Connector Architecture Attribute Descriptions

CHAPTER

12 Managing the WebLogic J2EE Connector Architecture

Based on the Sun Microsystems J2EE Connector Specification, Version 1.0, Proposed Final Draft 2, the WebLogic J2EE Connector Architecture integrates the J2EE Platform with one or more heterogeneous Enterprise Information Systems (EIS). The following sections explain how to manage and administer the WebLogic J2EE Connector Architecture:

- Overview of WebLogic J2EE Connectors
- Configuring Resource Adapters (Connectors) for Deployment
- Deploying Resource Adapters (Connectors)
- Viewing Deployed Resource Adapters (Connectors)
- Undeploying Deployed Resource Adapters (Connectors)
- Updating Deployed Resource Adapters (Connectors)
- Monitoring Connections
- Deleting a Connector
- Editing Resource Adapter Deployment Descriptors

For more information on BEA WebLogic J2EE Connectors, refer to Programming WebLogic J2EE Connectors.

Overview of WebLogic J2EE Connectors

BEA WebLogic Server continues to build upon the implementation of the Sun Microsystems J2EE Platform Specification, Version 1.3. The J2EE Connector Architecture adds simplified Enterprise Information System (EIS) integration to the J2EE platform. The goal is to leverage the strengths of the J2EE platform—including component models, and transaction and security infrastructures—to address the challenges of EIS integration.

The J2EE Connector Architecture provides a Java solution to the problem of connectivity between the multitude of application servers and EISes. By using the J2EE Connector Architecture, it is no longer necessary for EIS vendors to customize their product for each application server. An application server vendor (such as BEA WebLogic Server) that conforms to the J2EE Connector Architecture also does not need to add custom code whenever it wants to extend its application server to support connectivity to a new EIS.

The J2EE Connector Architecture enables an EIS vendor to provide a standard resource adapter (also referred to as a connector) for its EIS; the resource adapter plugs into an application server such as WebLogic Server and provides the underlying infrastructure for the integration between an EIS and the application server.

An application server vendor (BEA WebLogic Server) extends its system only once to support the J2EE Connector Architecture and is then assured of connectivity to multiple EISes. Likewise, an EIS vendor provides one standard resource adapter and it has the capability to plug in to any application server that supports the J2EE Connector Architecture.

Configuring Resource Adapters (Connectors) for Deployment

To configure a Connector using the WebLogic Server Administration Console:

1. Start the WebLogic Server Administration Console.

2. Select the Domain in which you will be working.
3. In the left pane of the Console, click Deployments.
4. In the left pane of the Console, click Connectors. A table is displayed in the right pane of the Console showing all the deployed Connectors.
5. Select the Configure a new Connector option.
6. Locate the archive file (RAR) to configure.

   Note: You can also configure an exploded application or component directory. Note that WebLogic Server deploys all components it finds in and below the specified directory.

7. Click [select] to the left of a directory or file to choose it and proceed to the next step.
8. Select a Target Server from among Available Servers.
9. Enter a name for the Connector in the provided field.
10. Click Configure and Deploy. The Console will display the Deploy panel, which lists deployment status and deployment activities for the Connector.
11. Using the available tabs, enter the following information:
    - Configuration—Edit the staging mode and enter the deployment order.
    - Targets—Indicate the Targets-Server for this configured Connector by moving the server from the Available list to the Chosen list.
    - Deploy—Deploy the Connector to all or selected targets or undeploy it from all or selected targets.
    - Monitoring—Enable session monitoring for the Connector.
    - Notes—Enter notes related to the Connector.

Configuring a Connector to Display a Connection Profile

The Administration Console can display call stacks for Connectors (connection profiles) as well as call stacks for leaked and idle connections. To configure a Connector to make this information available to the Administration Console, do the following:

1. After you deploy the Connector, in the left pane of the Administration Console, right click the Connector and choose Edit Connector Descriptor.
2. In the left pane, expand WebLogic RA and click Map Config Properties.
3. In the right pane, click Configure a new Map Config Property....
4. On the Configure a new Map Config Property, in the Description box, enter a description such as "Connection Profiling".
5. In the Config Property Name box, enter `connection-profiling-enabled`.
6. In the Config Property Value box, enter `true`.
7. Click Apply to save your changes.

For information on viewing this profile information, refer to "Monitoring Connections" on page 12-7.

Deploying Resource Adapters (Connectors)

To deploy a Connector using the WebLogic Server Administration Console:

1. Expand the Deployments node in the left pane.
2. Right-click on the Connectors node.
3. Select Configure a New Application.

4. Locate the archive file (RAR) to configure.

   Note: You can also configure an exploded application or component directory. Note that WebLogic Server deploys all components it finds in and below the specified directory.

5. Click [select] to the left of a directory or file to choose it and proceed to the next step.
6. Select a Target Server from among Available Servers.
7. Enter a name for the Connector in the provided field.
8. Click Configure and Deploy. The Console will display the Deploy panel, which lists deployment status and deployment activities for the Connector.
9. Use the Deploy button to deploy the Connector to all or selected targets or undeploy it from all or selected targets.
10. Test your Connector by accessing a resource through a Web browser. Access resources with a URL constructed as follows:

    http://myServerr:myPort/myConnector/resource

    Where:

    - myServer is the name of the machine hosting WebLogic Server.
    - myPort is the port number where WebLogic Server is listening for requests.
    - myConnector is the name of theConnector archive file (myConnector.rar, for instance) or the name of a directory containing the Connector.
    - resource is the name of a resource such as a JSP, HTTP servlet, or HTML page.

Viewing Deployed Resource Adapters (Connectors)

To view a deployed connector in the Administration Console:

1. In the Console, click Deployments.
2. Click the Connectors option.
3. View a list of deployed Connectors in the table displayed in the Console.

Undeploying Deployed Resource Adapters (Connectors)

To undeploy a deployed connector from the WebLogic Server Administration Console:

1. In the Console, click Deployments.
2. Click the Connectors option.
3. In the displayed table, click the name of the connector you wish to undeploy.
4. Click Apply.

Updating Deployed Resource Adapters (Connectors)

To update a deployed connector:

1. In the Console, click Deployments.

1. Click the Connectors option.
2. In the displayed table, click the name of the connector you wish to update.
3. Update the deployment status by accessing the Deploy tab.
4. Click Apply.

Monitoring Connections

The BEA J2EE Connector Architecture provides you with monitoring capabilities in the WebLogic Server Console that show detected leaks and provides a method for looking up stacks to determine which application(s) is causing the leak. Delete buttons in the Console allow you to dynamically close leaked connections that are identified; the option to delete connections is only available for connections that have exceeded their specified idle time and are safe to delete (in other words, the connection is not involved in a transaction).

The `connection-profiling-enabled` element of the `weblogic-ra.xml` file indicates whether or not the connection pool should store the call stacks of where each connection is allocated. If you set this element value to `true`, you can view this information on active connections through the Console. Also, you can view the stacks for leaked and idle connections, and you can debug components that fail to close connections.

Getting Started

There are two methods for accessing monitoring tools using the Console:

Method One

1. In the left pane of the Console, select Deployments > Connectors to display a list of connectors.
2. Right-click a connector, and select Monitor all Connector Connection Pool Runtimes from the pop-up menu.

   Connection pool run-time information is provided in the right pane for the selected connector.

Method Two

1. In the right pane of the Console, under Deployments, select Connectors.

   A connector table is displayed.
2. Under the Name column, click the connector to monitor.

3. In the Monitoring tab, select Monitor all Connector Connection Pool Runtimes.

   Connection pool run-time information is provided in the right pane for the selected connector.

Viewing Leaked Connections

A Connection Leak Profiles column in the Console allows you to view profile information pertaining to leaked connections. This column is not to be confused with the Leaked Connections Detected column, which simply displays the number of leaked connections.

A key difference between these two columns is the Connection Leak Profiles column is controlled by use of the `connection-profiling-enabled` setting in the `weblogic-ra.xml` file. By default, this setting is `false`, so normally the Connection Leak Profiles column will be zero (disabled). However, the Leaked Connections Detected column is always enabled and will always display the number of leaked connections.

There are two methods for viewing leaked connections using the Console:

Method One

1. In the left pane of the Console, select Deployments > Connectors to display a list of connectors.
2. Right-click a connector, and select View Leaked Connections from the pop-up menu.

   Connection pool run-time information for the selected connector is provided in the right pane.
3. Under the Connection Leak Profiles column, click the number of leaked connections pertaining to the selected connector.

   Leaked connection information is displayed in the right pane.

Method Two

1. In the right pane of the Console, under Deployments, select Connectors.

   A connector table is displayed.
2. Under the Name column, click the name of the connector to monitor.

3. In the Monitoring tab, select Monitor all Connector Connection Pool Runtimes.

   Connection pool run-time information for the selected connector is provided in the right pane.

4. Under the Connection Leak Profiles column, click the number of leaked connections pertaining to the selected connector.

   Leaked connection information is displayed in the right pane.

Viewing Idle Connections

A Connection Idle Profiles column in the Console allows you to view profile information pertaining to idle connections. This column is not to be confused with the Idle Connections Detected column, which simply displays the number of idle connections.

A key difference between these two columns is the Connection Idle Profiles column is controlled by use of the `connection-profiling-enabled` setting in the `weblogic-ra.xml` file. By default, this setting is `false`, so normally the Connection Idle Profiles column will be zero (disabled). However, the Idle Connections Detected column is always enabled and will always display the number of idle connections.

There are two methods for idle connections using the Console:

Method One

1. In the left pane of the Console, select Deployments > Connectors to display a list of connectors.

2. Right-click a connector, and select View Idle Connections from the pop-up menu.

   Connection pool run-time information for the selected connector is provided in the right pane.

3. Under the Connection Idle Profiles column, click the number of idle connections pertaining to the selected connector.

   Idle connection information is displayed in the right pane.

Method Two

1. In the right pane of the Console, under Deployments, select Connectors.

A connector table is displayed.

2. Under the Name column, click the name of the connector to monitor.
3. In the Monitoring tab, select Monitor all Connector Connection Pool Runtimes.

   Connection pool run-time information for the selected connector is provided in the right pane.
4. Under the Connection Idle Profiles column, click the number of idle connections pertaining to the selected connector.

   Idle connection information is displayed in the right pane.

Deleting Connections

This functionality is not currently implemented in the WebLogic Server Administration Console.

Deleting a Connector

To delete a connector, proceed as follows:

1. Select a connector to delete in the left pane of the Console under Deployments > Connectors > Connector Name.
2. In the table of connectors located in the right pane, select the Trash Can icon.

   The following message is displayed in the right pane:

   Are you sure you want to permanently delete <Connector Name> from the domain configuration?
3. Click Yes to delete the connector.

Editing Resource Adapter Deployment Descriptors

This section describes the procedure for editing the following resource adapter (connector) deployment descriptors using the Administration Console Deployment Descriptor Editor:

- ra.xml
- weblogic-ra.xml

For detailed information about the elements in the resource adapter deployment descriptors, refer to Programming WebLogic J2EE Connectors.

To edit the resource adapter deployment descriptors, follow these steps:

1. Invoke the Administration Console in your browser using the following URL:

   http://host:port/console

   where `host` refers to the name of the computer upon which WebLogic Server is running and `port` refers to the port number to which it is listening.

2. Click to expand the Deployments node in the left pane.

3. Click to expand the Connectors node under the Deployments node.

4. Right-click the name of the resource adapter whose deployment descriptors you want to edit and choose Edit Connector Descriptor from the drop-down menu.

   The Administration Console window appears in a new browser. The left pane contains a tree structure that lists all the elements in the two resource adapter deployment descriptors and the right pane contains a form for the descriptive elements of the `ra.xml` file.

5. To edit, delete, or add elements in the resource adapter deployment descriptors, click to expand the node in the left pane that corresponds to the deployment descriptor file you want to edit, as described in the following list:

   - The RA node contains the elements of the `ra.xml` deployment descriptor.

- The WebLogic RA node contains the elements of the `weblogic-ra.xml` deployment descriptor.

6. To edit an existing element in one of the resource adapter deployment descriptors, follow these steps:
    a. Navigate the tree in the left pane, clicking on parent elements until you find the element you want to edit.
    b. Click the element. A form appears in the right pane that lists either its attributes or subelements.
    c. Edit the text in the form in the right pane.
    d. Click Apply.
7. To add a new element to one of the resource adapter deployment descriptors, follow these steps:
    a. Navigate the tree in the left pane, clicking on parent elements until you find the name of the element you want to create.
    b. Right-click the element and chose Configure a New Element from the drop-down menu.
    c. Enter the element information in the form that appears in the right pane.
    d. Click Create.
8. To delete an existing element from one of the resource adapter deployment descriptors, follow these steps:
    a. Navigate the tree in the left pane, clicking on parent elements until you find the name of the element you want to delete.
    b. Right-click the element and chose Delete Element from the drop-down menu.
    c. Click Yes to confirm that you want to delete the element.
9. Once you have made all your changes to the resource adapter deployment descriptors, click the root element of the tree in the left pane. The root element is the either the name of the resource adapter `*.rar` archive file or the display name of the resource adapter.
10. Click Validate if you want to ensure that the entries in the resource adapter deployment descriptors are valid.

11. Click Persist to write your edits of the deployment descriptor files to disk in addition to WebLogic Server's memory.

13 Managing WebLogic Server Licenses

Your WebLogic Server requires a valid license to run. The following sections explain how to install and update WebLogic licenses:

- Installing a WebLogic Server License
- Updating a License

Installing a WebLogic Server License

An evaluation copy of WebLogic Server is enabled for 60 days so you can start using WebLogic Server immediately. To use WebLogic Server beyond the 60-day evaluation period, you will need to contact your salesperson about further evaluation or purchasing a license for each server (machine) on which you intend to use WebLogic Server. All WebLogic Server evaluation products are licensed for use on a single server with access allowed from up to five unique client IP addresses.

If you downloaded WebLogic Server from the BEA Web site, your evaluation license is included with the distribution. The WebLogic Server installation program allows you to specify the location of the BEA home directory, and installs a BEA license file, `license.bea`, in that directory.

Updating a License

You will need to update the BEA license file if one of the following is true:

- You have purchased additional BEA software.
- You obtain a new distribution that includes new products.
- You have applied for and received an extension of your 60-day evaluation license.

In any of these cases, you will receive a license update file by email as an attachment. To update your BEA license file, do the following:

1. Save the license update file under a name other than `license.bea` in the BEA home directory.
2. Make sure that the JDK (Java 2) is in your path. To add the JDK to your path, enter one of the following commands:
   - `set PATH=.\jdk131\bin;%PATH%` (Windows systems)
   - `set PATH=./jdk131/bin:$PATH` (UNIX systems)
3. In a command shell, enter the following command in the BEA home directory:

   ```
   UpdateLicense license_update_file
   ```

   where *`license_update_file`* is the name under which you saved the license update file that you received via email. Running this command updates the `license.bea` file.
4. Save a copy of your `license.bea` file in a safe place outside the WebLogic distribution. Although no one else can use your license file, you should save this information in a place protected from either malicious or innocent tampering.

A Using the WebLogic Java Utilities

WebLogic provides several Java programs that simplify installation and configuration tasks, provide services, and offer convenient shortcuts. The following sections describe each Java utility provided with WebLogic Server. The command-line syntax is specified for all utilities and, for some, examples are provided.

- showLicenses
- system
- verboseToZip
- version
- writeLicense

To use these utilities you must correctly set your `CLASSPATH`. For more information, see "Setting the Classpath."

AppletArchiver

The `AppletArchiver` utility runs an applet in a separate frame, keeps a record of all of the downloaded classes and resources used by the applet, and packages these into either a `.jar` file or a `.cab` file. (The `cabarc` utility is available from Microsoft.)

Syntax

```
$ java utils.applet.archiver.AppletArchiver URL filename
```

Argument	Definition
URL	URL for the applet.
filename	Local filename that is the destination for the `.jar/.cab` archive.

CertGen

The `CertGen` utility generates certificates that should only be used for demonstration or testing purposes and not in a production environment.

Syntax

```
$ java utils.CertGen password certfile keyfile [export]
```

Argument	Definition
password	Defines the password for the private key.
certfile	Defines the directory in which to copy the generated certificate file.
keyfile	Defines the directory in which to copy the generated private key file.
`export`	By default, the `CertGen` utility generates domestic strength certificates. Specify the [export] option if you want the tool to generate export strength certificates.

Example

To generate a certificate:

1. Copy the following files to the directory in which you run the CertGen tool:
   - `WL_HOME/server/lib/CertgenCA.der`—The certificate for a certificate authority trusted by WebLogic Server.
   - `WL_HOME/server/lib/CertGenCAKey.der`—The private key for a certificate authority trusted by WebLogic Server.
2. Enter the following command to generate certificate files named `testcert` with private key files named `testkey`:

```
$ java utils.CertGen mykeypass testcert testkey
Creating Domestic Key Strength - 1024

Encoding
................................................................
................................................................
................................................................
Created Private Key files - testkey.der and testkey.pem
```

```
Encoding
...........................................................
...........................................................
...........................................................
Created Certificate files - testcert.der and testcert.pem
...........................................................
```

ClientDeployer

You use `weblogic.ClientDeployer` to extract the client-side JAR file from a J2EE EAR file, creating a deployable JAR file. The `weblogic.ClientDeployer` class is executed on the Java command line with the following syntax:

```
java weblogic.ClientDeployer ear-file client
```

The *`ear-file`* argument is an expanded directory (or Java archive file with a `.ear` extension) that contains one or more client application JAR files.

For example:

```
java weblogic.ClientDeployer app.ear myclient
```

where `app.ear` is the EAR file that contains a J2EE client packaged in `myclient.jar`.

Once the client-side JAR file is extracted from the EAR file, use the `weblogic.j2eeclient.Main` utility to bootstrap the client-side application and point it to a WebLogic Server instance as follows:

java weblogic.j2eeclient.Main clientjar URL [application args]

For example

```
java weblogic.j2eeclient.Main helloWorld.jar
t3://localhost:7001 Greetings
```

Conversion

If you have used an earlier version of WebLogic, you must convert your `weblogic.properties` files. Instructions for converting your files using a conversion script are available in the Administration Console Online Help section called "Conversion."

der2pem

The der2pem utility converts an X509 certificate from DER format to PEM format. The .pem file is written in the same directory as the source .der file.

Syntax

```
$ java utils.der2pem derFile [headerFile] [footerFile]
```

Argument	Description
derFile	The name of the file to convert. The filename must end with a .der extension, and must contain a valid certificate in .der format.
headerFile	The header to place in the PEM file. The default header is "-----BEGIN CERTIFICATE-----". Use a header file if the DER file being converted is a private key file, and create the header file containing one of the following: ■ "-----BEGIN RSA PRIVATE KEY-----" for an unencrypted private key. ■ "-----BEGIN ENCRYPTED PRIVATE KEY-----" for an encrypted private key. **Note:** There must be a new line at the end of the header line in the file.
footerFile	The header to place in the PEM file. The default header is "-----END CERTIFICATE-----". Use a footer file if the DER file being converted is a private key file, and create the footer file containing one of the following in the header: ■ "-----END RSA PRIVATE KEY-----" for an unencrypted private key. ■ "-----END ENCRYPTED PRIVATE KEY-----" for an encrypted private key. **Note:** There must be a new line at the end of the header line in the file.

Example

```
$ java utils.der2pem graceland_org.der
Decoding
................................................................
```

dbping

The `dbping` command-line utility tests the connection between a DBMS and your client machine via a JDBC driver. You must complete the installation of the driver before attempting to use this utility. For more information on how to install a driver, see WebLogic jDrivers at http://e-docs.bea.com/wls/docs70/jdrivers.html.

Syntax

```
$ java -Dbea.home=license_location utils.dbping DBMS user password DB
```

Argument	Definition
license_location	The directory containing your WebLogic Server license (`license.bea`). For example, `d:\beaHome\`. Required only if using a BEA-supplied JDBC driver.
DBMS	Choose one of the following for your JDBC driver: WebLogic jDriver for Microsoft SQL Server: `MSSQLSERVER4` WebLogic jDriver for Oracle: `ORACLE` Oracle Thin Driver: `ORACLE_THIN` Sybase JConnect driver: `JCONNECT` Sybase JConnect 5.5 (JDBC 2.0) driver: `JCONN2`
user	Valid username for login. Use the same values you use with `isql` or `sqlplus`.
password	Valid password for the user. Use the same values you use with `isql` or `sqlplus`.

Argument	Definition

DB	Name of the database. Use the following format, depending on which JDBC driver you use:
	WebLogic jDriver for Microsoft SQL Server:
	`DBNAME@HOST:PORT`
	WebLogic jDriver for Oracle:
	`DBNAME`
	Oracle Thin Driver:
	`HOST:PORT:DBNAME`
	Sybase JConnect driver: `JCONNECT:`
	`HOST:PORT/DBNAME`
	Sybase JConnect driver: `JCONN2:`
	`HOST:PORT/DBNAME`
	
	Where:
	■ `HOST` is the name of the machine hosting the DBMS,
	■ `PORT` is port on the database host where the DBMS is listening for connections, and
	■ `DBNAME` is the name of a database on the DBMS. (For Oracle, this is the name of a DBMS defined in the `tnsnames.ora` file.)

Example

```
$ C:\bea\weblogic700b\samples\server\config\examples>java
utils.dbping ORACLE_THIN scott tiger lcdbsol1:1561:lcs901


**** Success!!! ****

You can connect to the database in your app using:

java.util.Properties props = new java.util.Properties();
props.put("user",     "scott");
props.put("password", "tiger");

java.sql.Driver d =
(java.sql.Driver)Class.forName("oracle.jdbc.driver.OracleD
river").newInstance();
java.sql.Connection conn =
d.connect("jdbc:oracle:thin:@lcdbsol1:1561:lcs901",
props);

// This mode is superior, especially in serverside classes because
// it avoids DriverManager calls are class synchronized, and will
// bottleneck any other JDBC in the server, even already-running
// connections, because all JDBC drivers useDriverManager.println()
// to log info and exceptions, and that call is also class
   synchronized.

// For repeated connecting, a single driver instance can be re-used.

**** or ****

Class.forName("oracle.jdbc.driver.OracleDriver").newInstance();
java.sql.Connection conn =
Driver.connect("jdbc:oracle:thin:@lcdbsol1:1561:lcs901", "scott",
"tiger");

**** or ****

java.util.Properties props = new java.util.Properties();
props.put("user",     "scott");
props.put("password", "tiger");
Class.forName("oracle.jdbc.driver.OracleDriver").newInstance();
java.sql.Connection conn =
Driver.connect("jdbc:oracle:thin:@lcdbsol1:1561:lcs901", props);
```

Deployer

weblogic.Deployer deploys J2EE applications and components to WebLogic Servers. For additional information, see Deployment Tools and Procedures at http://e-docs.bea.com/wls/docs70/programming/deploying.html#1094693.

The weblogic.Deployer utility is new in WebLogic Server 7.0, and replaces the earlier weblogic.deploy utility, which has been deprecated. For more information about the deprecated weblogic.deploy utility, see "Deploying Applications" in the WebLogic Server Administration Guide.

Syntax

```
% java weblogic.Deployer [options]
[-activate|-deactivate|-remove|-cancel|-list] [files]
```

Actions (select one of the following)

Action	Description
activate	Deploys or redeploys the application specified by -name to the servers specified by -targets.
cancel	Attempts to cancel the task identified by -id.
deactivate	Deactivates the application on the target servers. Deactivation suspends the deployed components, leaving staged data in place in anticipation of subsequent reactivation. This command only works in the two-phase deployment protocol.
delete_files	Removes files specified in the file list and leaves the application activated. This is valid only for unarchived applications. You must specify target servers.
deploy	A convenient alias for -activate.
examples	Displays example usages of the tool.
help	Prints a help message.
list	Lists the status of the task identified by -id.

Action	Description
remove	Physically removes the application and any staged data from the target servers. The components are deactivated and the targets are removed from the applications configuration. If you remove the application entirely, the associated MBeans are also deleted from the system configuration. This command only works with the two-phase deployment model.
undeploy	A convenient alias for `-unprepare`.
unprepare	Deactivates and unloads classes for the application identified by `-name` on the target servers, leaving the staged application files in a state where they may be edited or quickly reloaded.
upload	Transfers the specified source file(s) to the administration server. Use this option when you are on a remote system and want to deploy an application that resides on the remote system. The application files are uploaded to the WebLogic Server administration server prior to distribution to named target servers.
version	Prints version information.

Options

Option	Description
adminurl	https://<server>:<port> is the URL of the administration server. Default is http://localhost:7001.
debug	Turns on debug messages in the output log.
external_stage	Sets the `stagingMethod` attribute on the application Mbean when it is created so that the application will not be staged but the value of the staging path will be used when preparing the application.

Option	Description
id	The task identifier `-id` is a unique identifier for the deployment task. You can specify an `-id` with the `-activate`, `-deactivate`, or `-remove` commands, and use it later as an argument to `-cancel` or `-list`. Make sure the `-id` is unique from all other existing deployment tasks. The system generates an `-id` if you do not specify one.
name	The application `-name` specifies the name of the application being deployed. This can be the name of an existing, configured application or the name to use when creating a new configuration.
nostage	Sets the `no-staging` attribute on the ApplicationMBean, indicating that the application does not require staging. The system assumes the application already resides at the location specified by its Path attribute on the target servers.
nowait	Once the action is initiated, the tool prints the task id and exits. This is used to initiate multiple tasks and then monitor them later using the `-list` action.
password	Specifies the password on the command line. If you do not provide a password, you will be prompted for one.
remote	Signals that `weblogic.Deployer` is not running on the same machine as the administration server and that the source path should be passed through unchanged because it represents the path on the remote server.
source	Archive or directory, specifies the location of the file or directory to be deployed. Use this option to set the application Path. The source option should reference the root directory or archive being deployed. If using upload, the source path is relative to the current directory. Otherwise, it is relative to the administration server root directory—the directory where the `config.xml` file resides.

Option	Description
stage	Sets the `stagingMethod` attribute on the application when it is created so that the application will always be staged. This value overrides the `stagingMethod` attribute on any targeted servers.
targets	<server 1>,...<component>@<server N>, displays a comma-separated list of the server and/or cluster names. Each target may be qualified with a J2EE component name. This enables different components of the archive to deployed on different servers. When specified for an application that is already deployed, this list is an addition to the existing targets. If any existing targets are again specified, the application is redeployed on those targets and deployed on the new ones.
timeout	Seconds. Specifies the maximum time in seconds to wait for the completion of the deployment task. When the time expires, `weblogic.Deployer` prints out the current status of the deployment and exits.
user	User name.
verbose	Displays additional progress messages.

Examples

Examples of `weblogic.Deployer` commands:

- Deploying a New Application
- Redeploying an Application
- Redeploying Part of an Application
- Deactivating an Application
- Undeploying an Application
- Canceling a Deployment Task
- Listing All Deployment Tasks

Deploying a New Application

```
java weblogic.Deployer -adminurl http://admin:7001 -name app
-source /myapp/app.ear -targets server1,server2 -activate
```

Redeploying an Application

```
java weblogic.Deployer -adminurl http://admin:7001 -name app
-activate
```

Redeploying Part of an Application

```
java weblogic.Deployer -adminurl http://admin:7001 -name appname
-targets server1,server2 -activate jsps/*.jsp
```

Deactivating an Application

```
java weblogic.Deployer -adminurl http://admin:7001 -name app
-targets server1 -deactivate
```

Undeploying an Application

```
java weblogic.Deployer -adminurl http://admin:7001 -name app
-targets server -remove -id tag
```

Canceling a Deployment Task

```
java weblogic.Deployer -adminurl http://admin:7001 -cancel -id
tag
```

Listing All Deployment Tasks

```
java weblogic.Deployer -adminurl http://admin:7001 -list
```

EJBGen

EJBGen is an Enterprise JavaBeans 2.0 code generator. You can annotate your Bean class file with javadoc tags and then use EJBGen to generate the Remote and Home classes and the deployment descriptor files for an EJB application, reducing to one the number of EJB files you need to edit and maintain.

If you have installed BEA WebLogic 7.0 examples, see *SAMPLES_HOME\server\src\examples\ejb20\ejbgen\* for an example application that uses EJBGen.

For complete documentation of this tool, see EJBGen in WebLogic Server EJB Utilities at http://e-docs.bea.com/wls/docs70/ejb/EJB_utilities.html#1079050.

getProperty

The `getProperty` utility gives you details about your Java setup and your system. It takes no arguments.

Syntax

```
$ java utils.getProperty
```

Example

```
$ java utils.getProperty
-- listing properties --
user.language=en
java.home=c:\java11\bin\..
awt.toolkit=sun.awt.windows.WToolkit
file.encoding.pkg=sun.io
java.version=1.1_Final
file.separator=\
line.separator=
user.region=US
file.encoding=8859_1
java.vendor=Sun Microsystems Inc.
user.timezone=PST
user.name=mary
os.arch=x86
os.name=Windows NT
java.vendor.url=http://www.sun.com/
user.dir=C:\weblogic
java.class.path=c:\weblogic\classes;c:\java\lib\cla...
java.class.version=45.3
os.version=4.0
path.separator=;
user.home=C:\
```

ImportPrivateKey

The `ImportPrivateKey` utility is used to load a private key into a private keystore file.

Syntax

```
$ java utils.ImportPrivateKey keystore keystorepass alias keypass
certfile keyfile
```

Argument	Definition
keystore	Defines the name of the keystore file. A new keystore is created if one does not exist.
keystorepass	Defines the password to open the keystore file.
alias	Defines the name that is used to look up certificates and keys in the keystore.
keypass	Defines the password used to unlock the private key file and to protect the private key in the keystore.
certfile	The name of the certificate associated with the private key.
keyfile	The name of the file holding the protected private key.

Example

Use the following steps to:

- Generate a certificate and private key using the `CertGen` utility
- Create a keystore and store a private key using the `ImportPrivateKey` utility

1. Copy the `WL_HOME/server/lib/CertGenCA.der` file and the `WL_HOME/server/lib/CertGenCAkey.der` file to your working directory.
2. Use the `utils.CertGen` utility to generate a certificate and private key. See Using the CertGen Tool at http://e-docs.bea.com/wls/docs70/secmanage/ssl.html#1165276.

```
java utils.CertGen mykeypass testcert testkey
Creating Domestic Key Strength - 1024

Encoding
...............................................................
...............................................................
...............................................................
Created Private Key files - testkey.der and testkey.pem
Encoding
...............................................................
...............................................................
...............................................................
Created Certificate files - testcert.der and testcert.pem
...............................................................
```

3. Convert the certificate from DER format to PEM format.

```
D:\bea2\weblogic700\samples\server\src>java utils.der2pem
CertGenCA.der
Encoding
...............................................................
...............................................................
```

4. Concatenate the certificate and the Certificate Authority (CA).

```
D:\bea2\weblogic700\samples\server\src>type testcert.pem
CertGenCA.pem >> newcerts.pem
```

5. Create a new keystore named `mykeystore` and load the private key located in the `testkey.pem` file.

```
D:\bea2\weblogic700\samples\server\src>java utils.ImportPrivateKey
mykeystore mypasswd mykey mykeypass newcerts.pem testkey.pem
Keystore file not found, creating it
```

logToZip

The `logToZip` utility searches an HTTP server log file in common log format, finds the Java classes loaded into it by the server, and creates an uncompressed `.zip` file that contains those Java classes. It is executed from the document root directory of your HTTP server.

To use this utility, you must have access to the log files created by the HTTP server.

Syntax

```
$ java utils.logToZip logfile codebase zipfile
```

Argument	Definition
logfile	Required. Fully-qualified pathname of the log file.
codebase	Required. Code base for the applet, or `""` if there is no code base. By concatenating the code base with the full package name of the applet, you get the full pathname of the applet (relative to the HTTP document root).
zipfile	Required. Name of the `.zip` file to create. The resulting `.zip` file is created in the directory in which you run the program. The pathname for the specified file can be relative or absolute. In the examples, a relative pathname is given, so the `.zip` file is created in the current directory.

Examples

The following example shows how a `.zip` file is created for an applet that resides in the document root itself, that is, with no code base:

```
$ cd /HTTP/Serv/docs
$ java utils.logToZip /HTTP/Serv/logs/access "" app2.zip
```

The following example shows how a `.zip` file is created for an applet that resides in a subdirectory of the document root:

```
C:\>cd \HTTP\Serv
C:\HTTP\Serv>java utils.logToZip \logs\applets\classes app3.zip
```

MulticastTest

The `MulticastTest` utility helps you debug multicast problems when configuring a WebLogic Cluster. The utility sends out multicast packets and returns information about how effectively multicast is working on your network. Specifically, `MulticastTest` displays the following types of information via standard output:

1. A confirmation and sequence ID for each message sent out by this server.
2. The sequence and sender ID of each message received from any clustered server, including this server.
3. A missed-sequenced warning when a message is received out of sequence.
4. A missed-message warning when an expected message is not received.

To use `MulticastTest`, start one copy of the utility on each node on which you want to test multicast traffic.

Warning: Do NOT run the `MulticastTest` utility by specifying the same multicast address (the `-a` parameter) as that of a currently running WebLogic Cluster. The utility is intended to verify that multicast is functioning properly before starting your clustered WebLogic Servers.

For information about setting up multicast, see the configuration documentation for the operating system/hardware of the WebLogic Server host. For more information about configuring a cluster, see *Using WebLogic Server Clusters*.

Syntax

```
$ java utils.MulticastTest -n name -a address [-p portnumber]
   [-t timeout] [-s send]
```

Argument	Definition
`-n name`	Required. A name that identifies the sender of the sequenced messages. Use a different name for each test process you start.
`-a address`	Required. The multicast address on which: (a) the sequenced messages should be broadcast; and (b) the servers in the clusters are communicating with each other. (The default for any cluster for which a multicast address is not set is 237.0.0.1.)

Argument	Definition
`-p` *portnumber*	Optional. The multicast port on which all the servers in the cluster are communicating. (The multicast port is the same as the listen port set for WebLogic Server, which defaults to 7001 if unset.)
`-t` *timeout*	Optional. Idle timeout, in seconds, if no multicast messages are received. If unset, the default is 600 seconds (10 minutes). If a timeout is exceeded, a positive confirmation of the timeout is sent to `stdout`.
`-s` *send*	Optional. Interval, in seconds, between sends. If unset, the default is 2 seconds. A positive confirmation of each message sent out is sent to `stdout`.

Example

```
$ java utils.MulticastTest -N server100 -A 237.155.155.1
Set up to send and receive on Multicast on Address 237.155.155.1 on
port 7001
Will send a sequenced message under the name server100 every 2
seconds.
Received message 506 from server100
Received message 533 from server200
   I (server100) sent message num 507
Received message 507 from server100
Received message 534 from server200
   I (server100) sent message num 508
Received message 508 from server100
Received message 535 from server200
   I (server100) sent message num 509
Received message 509 from server100
Received message 536 from server200
   I (server100) sent message num 510
Received message 510 from server100
Received message 537 from server200
   I (server100) sent message num 511
Received message 511 from server100
Received message 538 from server200
   I (server100) sent message num 512
Received message 512 from server100
Received message 539 from server200
   I (server100) sent message num 513
Received message 513 from server100
```

myip

The `myip` utility returns the IP address of the host.

Syntax

```
$ java utils.myip
```

Example

```
$ java utils.myip
Host toyboat.toybox.com is assigned IP address: 192.0.0.1
```

pem2der

The `pem2der` utility converts an X509 certificate from PEM format to DER format. The `.der` file is written in the same directory as the source `.pem` file.

Syntax

```
$ java utils.pem2der pemFile
```

Argument	Description
pemFile	The name of the file to be converted. The filename must end with a `.pem` extension, and it must contain a valid certificate in `.pem` format.

Example

```
$ java utils.pem2der graceland_org.pem
Decoding
.................................................................
.................................................................
.................................................................
.................................................................
.................................................................
```

Schema

The `Schema` utility lets you upload SQL statements to a database using the WebLogic JDBC drivers. For additional information about database connections, see *Programming WebLogic JDBC*.

Syntax

```
$ java utils.Schema driverURL driverClass [-u username]
   [-p password] [-verbose] SQLfile
```

Argument	Definition
driverURL	Required. URL for the JDBC driver.
driverClass	Required. Pathname of the JDBC driver class.
-u *username*	Optional. Valid username.
-p *password*	Optional. Valid password for the user.
-verbose	Optional. Prints SQL statements and database messages.
SQLfile	Required. Text file with SQL statements.

Example

The following code shows a `Schema` command line for the `examples.utils` package:

```
D:\bea\weblogic700\samples\server\src>java utils.Schema
"jdbc:pointbase:server://localhost/demo"
"com.pointbase.jdbc.jdbcUniversalDriver" -u "examples"
-p "examples" examples/utils/ddl/demo.ddl

utils.Schema will use these parameters:

url: jdbc:pointbase:server://localhost/demo
driver: com.pointbase.jdbc.jdbcUniversalDriver
dbserver: null
user: examples
password: examples
SQL file: examples/utils/ddl/demo.ddl
```

showLicenses

The `showLicenses` utility displays license information about BEA products installed in this machine.

Syntax

```
$ java -Dbea.home=license_location utils.showLicenses
```

Argument	Description
license_location	The fully qualified name of the directory where the `license.bea` file exists.

Example

```
$ java -Dbea.home=d:\bea utils.showLicense
```

system

The `system` utility displays basic information about your computer's operating environment, including the manufacturer and version of your JDK, your `CLASSPATH`, and details about your operating system.

Syntax

```
$ java utils.system
```

Example

```
$ java utils.system
* * * * * * * java.version * * * * * * *
1.1.6

* * * * * * * java.vendor * * * * * * *
Sun Microsystems Inc.

* * * * * * * java.class.path * * * * * * *
\java\lib\classes.zip;\weblogic\classes;
\weblogic\lib\weblogicaux.jar;\weblogic\license
...

* * * * * * * os.name * * * * * * *
Windows NT

* * * * * * * os.arch * * * * * * *
x86

* * * * * * * os.version * * * * * * *
4.0
```

verboseToZip

When executed from the document root directory of your HTTP server, `verboseToZip` takes the standard output from a Java application run in verbose mode, finds the Java classes referenced, and creates an uncompressed `.zip` file that contains those Java classes.

Syntax

```
$ java utils.verboseToZip inputFile zipFileToCreate
```

Argument	Definition
inputFile	Required. Temporary file that contains the output of the application running in verbose mode.
zipFileToCreate	Required. Name of the `.zip` file to be created. The resulting `.zip` file is be created in the directory in which you run the program.

UNIX Example

```
$ java -verbose myapplication > & classList.tmp
$ java utils.verboseToZip classList.tmp app2.zip
```

NT Example

```
$ java -verbose myapplication > classList.tmp
$ java utils.verboseToZip classList.tmp app3.zip
```

version

The `version` utility displays version information about your installed WebLogic via `stdout`.

Syntax

```
$ java weblogic.Admin -url host:port -username username -password
password VERSION
```

Example

```
$ java weblogic.Admin
-url localhost:7001 -username system -password foo VERSION
```

writeLicense

The `writeLicense` utility writes information about all your WebLogic licenses in a file called `writeLicense.txt`, located in the current directory. This file can then be emailed, for example, to WebLogic technical support.

Syntax

```
$ java utils.writeLicense -nowrite -Dweblogic.system.home=path
```

Argument	Definition
`-nowrite`	Required. Sends the output to `stdout` instead of `writeLicense.txt`.
`-Dweblogic.system.home`	Required. Sets WebLogic system home (the root directory of your WebLogic installation). This argument is required unless you are running `writeLicense` from your WebLogic system home.

Examples

```
$ java utils.writeLicense -nowrite
```

Example of UNIX Output

```
* * * * * * System properties * * * * * *

* * * * * * * java.version * * * * * * *
1.1.7

* * * * * * * java.vendor * * * * * * *
Sun Microsystems Inc.

* * * * * * * java.class.path * * * * * * *
c:\weblogic\classes;c:\weblogic\lib\weblogicaux.jar;
c:\java117\lib\classes.zip;c:\weblogic\license
...
```

Example of Windows NT Output

```
* * * * * * * os.name * * * * * * *
Windows NT

* * * * * * * os.arch * * * * * * *
x86

* * * * * * * os.version * * * * * * *
4.0

* * * * * * IP * * * * * *
Host myserver is assigned IP address: 192.1.1.0

* * * * * * Location of WebLogic license files * * * * * *
No WebLogicLicense.class found

No license.bea license found in
weblogic.system.home or current directory

Found in the classpath: c:/weblogic/license/license.bea
Last Modified: 06/02/1999 at 12:32:12

* * * * * * Valid license keys * * * * * *
Contents:
Product Name    : WebLogic
 IP Address     : 192.1.1.0-255
 Expiration Date: never
 Units          : unlimited
 key            : b2fcf3a8b8d6839d4a252b1781513b9
...

* * * * * * All license keys * * * * * *
Contents:
Product Name    : WebLogic
 IP Address     : 192.1.1.0-255
 Expiration Date: never
 Units          : unlimited
 key            : b2fcf3a8b8d6839d4a252b1781513b9
...

* * * * * * WebLogic version * * * * * *
WebLogic Build: 4.0.x xx/xx/1999 10:34:35 #xxxxx
```

B WebLogic Server Command-Line Interface Reference

The following sections discuss the WebLogic Server command-line interface syntax, and describe each WebLogic Server administration, connection pool administration, and MBean management command:

To deploy J2EE modules on a WebLogic Server instance, use the `weblogic.Deployer` command-line utility. See "Deployment Tools and Procedures" in *Deploying WebLogic Server Applications*.

About the Command-Line Interface

As an alternative to the Administration Console, WebLogic Server offers a command-line interface to its administration tools, as well as to many configuration and run-time MBean properties.

Use the command-line interface if:

- You want to create scripts for administration and management efficiency.
- You cannot access the Administration Console through a browser.
- You prefer using the command-line interface over a graphical user interface.

Before You Begin

Before you use the `weblogic.Admin` utility, set up your environment as follows:

1. Install and configure the WebLogic Server software, as described in the *WebLogic Server Installation Guide*. See `http://e-docs.bea.com/wls/docs70/install/index.html`.
2. Add WebLogic Server classes to the `CLASSPATH` environment variable. See "Setting the Classpath" on page 2-18.
3. If you want the `weblogic.Admin` utility to use a listen port that is reserved for administration traffic, you must configure a domain-wide administration port as described in "Configuring a Domain-Wide Administration Port" at http://e-docs.bea.com/wls/docs70/admin_domain/network.html#1126818.

Using WebLogic Server Administration Commands

This section presents the syntax and required arguments for using WebLogic Server administration commands. The commands are not case-sensitive.

Syntax

```
java weblogic.Admin [-url URL]
   [ { -username username [-password password] } |
     { [-userconfigfile config-file] [-userkeyfile admin-key] }
   ]
   COMMAND arguments
```

Connection and User Credentials Arguments

When you invoke most `weblogic.Admin` commands, you specify the arguments in Table 13-1 to connect to a WebLogic Server instance and to specify the user credentials of a WebLogic Server user who has permission to invoke the command.

Table 13-1 Conection and User Credentials Arguments

Argument	Definition
`-url` *`URL`*	Specifies one of the following: ■ The listen address of the domain's Administration Server. In most cases, we recommend that you use this URL because it runs the command within the security context of the Administration Server. ■ The listen address of the WebLogic Server that is the target of the command. Use this URL if you cannot access the Administration Server and you want to target a Managed Server. The format is *`hostname`*`:`*`port`*. The default is `localhost:7001`. **Note:** If you want to specify an administration port, make sure your system administrator has set up an administration port for **all** server instances in the domain as described in "Configuring a Domain-Wide Administration Port" in the *Creating and Configuring WebLogic Server Domains* guide.
`-username` *`username`*	Username that has permission to invoke the command you specify. If you do not specify this argument, `weblogic.Admin` uses a user-configuration file and key file. For information about permissions for system administration tasks, refer to "Protecting System Administration Operations" on page 3-1.
`-password` *`password`*	The password that is associated with the username. If you specify `-username` *`username`* but do not specify the `-password` argument, `weblogic.Admin` prompts you for a password. If *`WL_HOME`*`\server\bin` is specified in the `PATH` environment variable, `weblogic.Admin` uses a set of WebLogic Server libraries that prevent the password from being echoed to standard out. For information on setting environment variables, see "Setting the Classpath" on page 2-18.

Table 13-1 Conection and User Credentials Arguments

Argument	Definition
`-userconfigfile` *`config-file`*	Specifies the name and location of a user-configuration file, which contains an encrypted username and password. The encrypted username must have permission to invoke the command you specify. If you do not specify `-userconfigfile` *`config-file`*, `weblogic.Admin` searches for a user-configuration file at the default path name. (See "STOREUSERCONFIG" on page B-30.)
`-userkeyfile` *`admin-key`*	Specifies the name and location of the key file that is associated with the user-configuration file you specify. When you create a user-configuration file, the `STOREUSERCONFIG` command uses a key file to encrypt the username and password. Only the key file that encrypts a user-configuration file can decrypt the username and password. If you do not specify `-userkeyfile` *`admin-key`*, `weblogic.Admin` searches for a key file at the default path name. (See "STOREUSERCONFIG" on page B-30.)

Note: The exit code for all commands is 1 if the Administration client cannot connect to the server.

Summary of User Credentials Arguments

Table 13-1 describes the alternatives that the `weblogic.Admin` utility provides for passing usernames and passwords to a server instance.

In a development environment in which security is not a top priority, you can use the `-username` and `-password` arguments when invoking the `weblogic.Admin` utility directly on the command line or in scripts. With these arguments, the username and password are not encrypted. If you store the values in a script, the user credentials can be used by anyone who has read access to the script.

In an environment in which security is a top priority, create user-configuration files and key files. A user-configuration file contains encrypted user credentials that can be decrypted only by a single key file. You can include the `-userconfigfile` *`config-file`* and `-userkeyfile` *`admin-key`* arguments in scripts without exposing the plain text user credentials to those with read privileges for the script. For information about creating a user-configuration and key file, see "STOREUSERCONFIG" on page B-30.

The following list summarizes the order of precedence for the `weblogic.Admin` user-credentials arguments:

- If you specify `-username` *`username`* `-password` *`password`*, the utility passes the unencrypted values to the server instance you specify in the `-url` argument.

  These arguments take precedence over the `{ -userconfigfile` *`config-file`* `-userkeyfile` *`admin-key`* `}` arguments. If you specify both `{ -username` *`username`* `-password` *`password`* `}` and `{ -userconfigfile` *`config-file`* `-userkeyfile` *`admin-key`* `}`, the `weblogic.Admin` utility uses the `{ -username` *`username`* `-password` *`password`* `}` arguments and ignores the user-configuration and key file arguments.

- If you specify `-username` *`username`*, the utility prompts for a password. Then it passes the unencrypted values to the server instance you specify in the `-url` argument.

  This argument also takes precedence over the `{ -userconfigfile` *`config-file`* `-userkeyfile` *`admin-key`* `}` arguments.

- If you specify `{ -userconfigfile` *`config-file`* `-userkeyfile` *`admin-key`* `}` and do not specify `{ -username` *`username`* `[-password` *`password`*`] }`, the utility passes the values that are encrypted in *`config-file`* to the server instance you specify in the `-url` argument.

- If you specify neither `{ -username` *`username`* `[-password` *`password`*`] }` nor `{ -userconfigfile` *`config-file`* `-userkeyfile` *`admin-key`* `}`, the utility searches for a user-configuration file and key file at the default path names. The default path names vary depending on the JVM and the operating system. See "Configuring the Default Path Name" on page B-32.

Examples of Providing User Credentials

The following command specifies the username **weblogic** and password **weblogic** directly on the command line:

```
java weblogic.Admin -username weblogic -password weblogic COMMAND
```

The following command uses a user-configuration file and key file that are located at the default pathname:

```
java weblogic.Admin COMMAND
```

See "Configuring the Default Path Name" on page B-32.

The following command uses a user-configuration file named `c:\wlUser1-WebLogicConfig.properties` and a key file named `e:\secure\myKey`:

```
java -userconfigfile c:\wlUser1-WebLogicConfig.properties
-userkeyfile e:\secure\myKey COMMAND
```

WebLogic Server Administration Command Reference

The following sections provide information about the WebLogic server administration commands.

Table B-1 presents an overview of WebLogic Server administration commands. The following sections describe command syntax and arguments, and provide an example for each command.

See also "WebLogic Server Connection Pools Administration Command Reference" on page B-39.

Table B-1 WebLogic Server Administration Commands Overview

Task	Command	Description
Cancel shutting down a WebLogic Server	CANCEL_SHUTDOWN	(Deprecated) Cancels the `SHUTDOWN` command for the WebLogic Server that is specified in the URL. See "CANCEL_SHUTDOWN" on page B-10.
Connect to WebLogic Server	CONNECT	Makes the specified number of connections to the WebLogic Server and returns two numbers representing the total time for each round trip and the average amount of time (in milliseconds) that each connection is maintained. See "CONNECT" on page B-11.

Table B-1 WebLogic Server Administration Commands Overview (Continued)

Task	Command	Description
Force shutdown of a WebLogic Server	FORCESHUTDOWN	Immediately terminates a WebLogic Server process. See "FORCESHUTDOWN" on page B-12.
Determine the current state of a server	GETSTATE	Returns the current state of the WebLogic Server. See "GETSTATE" on page B-14.
Get Help for one or more commands	HELP	Provides syntax and usage information for all WebLogic Server commands (by default) or for a single command if a command value is specified on the HELP command line. See "HELP" on page B-15.
View WebLogic Server licenses	LICENSES	Lists the licenses for all the WebLogic Server instances installed on a specific server. See "LICENSES" on page B-16.
List JNDI naming tree node bindings	LIST	Lists the bindings of a node in the JNDI naming tree. See "LIST" on page B-17.
Lock WebLogic Server	LOCK	(Deprecated) Locks a WebLogic Server against non-privileged logins. Any subsequent login attempt initiates a security exception which may contain an optional string message. See "LOCK" on page B-18.
Migrate Services	MIGRATE	Migrates a JMS service or a JTA service to a targeted server within the cluster. See "MIGRATE" on page B-19
Verify WebLogic Server listening ports	PING	Sends a message to verify that a WebLogic Server is listening on a port, and is ready to accept WebLogic client requests. See "PING" on page B-21.
Move a server from the `STANDBY` state to `RUNNING`	RESUME	Makes a server available to receive requests from external clients. See "RESUME" on page B-22.
Viewing server log files	SERVERLOG	Displays the server log file generated on a specific server. See "SERVERLOG" on page B-23.
Shut down a WebLogic Server	SHUTDOWN	Shuts down a WebLogic Server. See "SHUTDOWN" on page B-24.

Table B-1 WebLogic Server Administration Commands Overview (Continued)

Task	Command	Description
Start a remote managed WebLogic Server	START	Uses a configured Node Manager to start a Managed Server in the `RUNNING` state. See "START" on page B-26.
Start a remote managed WebLogic Server and place it in the `STANDBY` state	STARTINSTANDBY	Uses a configured Node Manager to start a Managed Server and place it in the `STANDBY` state. See "STARTINSTANDBY" on page B-28.
Encrypt user credentials in a file	STOREUSERCONFIG	Creates an encrypted user-configuration file and its associated key file. You can pass the encrypted values to a server instance instead of entering a username and password on the command line. See "STOREUSERCONFIG" on page B-30.
View threads	THREAD_DUMP	Provides a real-time snapshot of the WebLogic Server threads that are currently running. See "THREAD_DUMP" on page B-36.
Unlock a WebLogic Server	UNLOCK	(Deprecated) Unlocks the specified WebLogic Server after a `LOCK` operation. See "UNLOCK" on page B-37.
View WebLogic Server version	VERSION	Displays the version of the WebLogic Server software that is running on the machine specified by the value of *`URL`*. See "VERSION" on page B-38.

Note: The exit code for all commands is 1 if the Administration client cannot connect to the server.

CANCEL_SHUTDOWN

(Deprecated) The CANCEL_SHUTDOWN command cancels the SHUTDOWN command for a specified WebLogic Server.

When you use the SHUT_DOWN command, you can specify a delay (in seconds). An administrator may cancel the shutdown command during the delay period. Be aware that the SHUTDOWN command disables logins, and they remain disabled even after cancelling the shutdown. Use the UNLOCK command to re-enable logins.

See "SHUTDOWN" on page B-24 and "UNLOCK" on page B-37.

This command is deprecated because the ability to specify a delay in the `SHUTDOWN` command is also deprecated. Instead of specifying a delay in the `SHUTDOWN` command, you can now set attributes to control how a server shuts down. For more information, refer to "Setting the Timeout Period for Forced Shutdown Operations" in the Administration Console Online Help.

Syntax

```
java weblogic.Admin [Connection and User Credentials Arguments]
CANCEL_SHUTDOWN
```

Example

In the following example, a system user named `weblogic` with a password of `weblogic` requests to cancel the shutdown of the WebLogic Server listening on port 7001 on machine `localhost`:

```
java weblogic.Admin -url t3://localhost:7001 -username weblogic
   -password weblogic CANCEL_SHUTDOWN
```

CONNECT

Makes the specified number of connections to the WebLogic Server and returns two numbers representing the total time for each round trip and the average amount of time (in milliseconds) that each connection is maintained.

Syntax

```
java weblogic.Admin [Connection and User Credentials Arguments]
CONNECT count
```

Argument	Definition
count	Number of connections to be made.

Example

In the following example, a user with the name `adminuser` and the password `gumby1234` runs the `CONNECT` command to establish 25 connections to a server named `localhost` and return information about those connections:

```
java weblogic.Admin -url localhost:7001 -username adminuser
   -password gumby1234 CONNECT 25
```

FORCESHUTDOWN

Immediately terminates a server process.

When you issue this command, the server notifies all applications and subsystems to drop all current work and release all resources. A forceful shutdown could result in rolled back transactions and session loss for some clients. You can shut down a server forcefully from any state.

If you use this command for a Managed Server and it fails to respond, and if you started the server with the Node Manager, the Node Manager kills the server process.

For information on performing this task from the Administration Console, refer to Forcing Shutdown of a Server in the Administration Console Online Help.

This command is affected by the timeout period for LifeCycle operations. For more information, refer to Timeout Period for LifeCycle Operations in the *WebLogic Administration Guide* and Setting the Timeout Period for LifeCycle Operations in the Administration Console Online Help.

Syntax

```
java weblogic.Admin [Connection and User Credentials Arguments]
FORCESHUTDOWN [targetserver]
```

Argument	Definition
targetserver	Optional. The name of the server to shut down. If you do not specify a value, the command shuts down the server that you specified in the `-url` argument.

Example

In the following example, a user with the username `adminuser` and password `gumby1234` forces a server named MyServer to shut down via the Administration Server:

```
java weblogic.Admin -url myAdminServer:7001 -username adminuser
-password gumby1234 FORCESHUTDOWN MyServer
```

In the following example, the Administration Server is not available. The same user logs on to a Managed Server's host machine and issues the following command:

```
java weblogic.Admin -url localhost:7001 -username adminuser
-password gumby1234 FORCESHUTDOWN
```

GETSTATE

Returns the current state of a server.

For more information about server states, refer to The Server Lifecycle.

Syntax

```
java weblogic.Admin [Connection and User Credentials Arguments]
GETSTATE targetserver
```

Argument	Definition
`targetserver`	Optional. The name of the server to shut down. If you do not specify a value, the command returns the state of the server that you specified in the `-url` argument.

Example

In the following example, a user with the `adminuser` username and password `gumby1234` attempts to determine the state of a server named MyServer via the Administration Server:

```
java weblogic.Admin -url myAdminServer:7001 -username adminuser
-password gumby1234 GETSTATE MyServer
```

HELP

Provides syntax and usage information for all WebLogic Server commands (by default) or for a single command if a command value is specified on the HELP command line.

Syntax

```
java weblogic.Admin HELP [COMMAND]
```

Example

In the following example, information about using the `PING` command is requested:

```
java weblogic.Admin HELP PING
```

The `HELP` command returns the following to `stdout`:

```
Usage: weblogic.Admin [-url url] -username username
   [-password password] <COMMAND> <ARGUMENTS>

         PING <count> <bytes>
```

LICENSES

Lists the licenses for all WebLogic Server instances installed on the specified server.

Syntax

```
java weblogic.Admin [Connection and User Credentials Arguments]
LICENSES
```

Example

In the following example, an administrator using the default username (`installadministrator`) and default password (`installadministrator`) requests the license information for a WebLogic Server running on port 7001 of machine `localhost`:

```
java weblogic.Admin -url localhost:7001 -username
installadministrator
   -password installadministrator LICENSES
```

LIST

Lists the bindings of a node in the JNDI naming tree.

Syntax

```
java weblogic.Admin [Connection and User Credentials Arguments]
LIST context
```

Argument	Definition
context	Required. The JNDI context for lookup, for example, `weblogic`, `weblogic.ejb`, `javax`.

Example

In this example, user `adminuser`, who has a password of `gumby1234`, requests a list of the node bindings in `weblogic.ejb`:

```
java weblogic.Admin -url localhost:7001 -username adminuser
-password gumby1234 LIST weblogic.ejb
```

LOCK

(Deprecated) Locks a WebLogic Server against non-privileged logins. Any subsequent login attempt initiates a security exception which may contain an optional string message.

Note: This command is privileged. It requires the password for the WebLogic Server administrative user.

Instead of using the `LOCK` command, start a server in the `STANDBY` state. In this state, a server instance responds only to administrative requests over the domain-wide administration port. For more information, refer to "STARTINSTANDBY" on page B-28.

Syntax

```
java weblogic.Admin [Connection and User Credentials Arguments]
LOCK "string_message"
```

Argument	Definition
"string_message"	Optional. Message, in double quotes, to be supplied in the security exception that is thrown if a non-privileged user attempts to log in while the WebLogic Server is locked.

Example

In the following example, a WebLogic Server is locked.

```
java weblogic.Admin -url localhost:7001 -username adminuser
   -password gumby1234
   LOCK "Sorry, WebLogic Server is temporarily out of service."
```

Any application that subsequently tries to log into the locked server with a non-privileged username and password receives the specified message: `Sorry, WebLogic Server is temporarily out of service.`

MIGRATE

Migrates a JMS service or a JTA Transaction Recovery service to a targeted server within a server cluster.

Syntax

To migrate JMS resources:

```
java weblogic.Admin [Connection and User Credentials Arguments]
  MIGRATE -migratabletarget "serverName (migratable)"
  -destination serverName [-sourcedown] [-destinationdown]
```

To migrate JTA resources:

```
java weblogic.Admin [-url URL]
  -username username [-password password]
  MIGRATE -jta -migratabletarget serverName
  -destination serverName [-sourcedown] [-destinationdown]
```

Argument	Definition
`{-url [protocol://]listen-address:listen-port}`	Specify the listen address and listen port of the Administration Server. If you specify a secure listen port, you must also specify a secure protocol. If you do not specify a value, the command assumes `t3://localhost:7001`.
`-jta`	Specifies that the migration is a migration of JTA services. If you do not specify this argument, the `MIGRATE` command migrates JMS services.
`-migratabletarget`	Names the server from which the service will migrate. The syntax for the server name varies depending on the type of service you are migrating: ■ For JMS, specify "*servername* (migratable)" For example, "myserver (migratable)" ■ For JTA, specify *servername* For example, `myserver`
`-destination`	Names the server to which the service will migrate.

Argument	Definition
`-sourcedown`	Specifies that the source server is down. This switch should be used very carefully. If the source server is not in fact down, but only unavailable because of network problems, the service will be activated on the destination server without being removed from the source server, resulting in two simultaneous running versions of the same service, *which could cause corruption of the transaction log or of JMS messages*.
`-destinationdown`	Specifies that the destination server is down. A JMS service migrated to a non-running server will be lost. When migrating the JTA Transaction Recovery Service to a non-running server, the target server will assume recovery services when it is started.

Examples

In the following example, a JMS service is migrated from myserver2 to myserver3.

```
java weblogic.Admin -url AdminHost:7001 -username adminuser
   -password gumby1234 MIGRATE
   -migratabletarget "myserver2 (migratable)"
   -destination myserver3
```

In the following example, a JTA Transaction Recovery service is migrated from myserver2 to myserver3.

```
java weblogic.Admin -url AdminHost:7001 -username adminuser
   -password gumby1234 MIGRATE -jta
   -migratabletarget myserver2 -destination myserver3 -sourcedown
```

PING

Sends a message to verify that a WebLogic Server is listening on a port, and is ready to accept WebLogic client requests.

Syntax

```
java weblogic.Admin [Connection and User Credentials Arguments]
PING [round_trips] [message_length]
```

Argument	Definition
round_trips	Optional. Number of pings.
message_length	Optional. Size of the packet to be sent in each ping. Requests for pings with packets larger than 10 MB throw exceptions.

Example

In the following example, the command checks a WebLogic Server running on port 7001 of machine `localhost` ten (10) times.

```
java weblogic.Admin -url localhost:7001 -username adminuser
   -password gumby1234 PING 10
```

RESUME

Moves a server from the STANDBY state to the RUNNING state.

For information on performing this task from the Administration Console, refer to Resuming a Server in the Administration Console Online Help. For more information about server states, refer to The Server Lifecycle.

Syntax

```
java weblogic.Admin [Connection and User Credentials Arguments]
RESUME [targetserver]
```

Argument	Definition
targetserver	Optional. The name of the server to shut down. If you do not specify a value, the command resumes the server that you specified in the `-url` argument.

Example

In the following example, a user with the `adminuser` username and password `gumby1234` attempts to resume a server named MyServer via the Administration Server:

```
java weblogic.Admin -url myAdminServer:7001 -username adminuser
-password gumby1234 RESUME MyServer
```

SERVERLOG

Displays the log file generated on a specific server.

- If you do not specify a URL, the server log for the Administration Server is displayed by default.
- If you specify a server URL, you can retrieve a log for a non-Administration Server.
- If you omit the `starttime` and `endtime` arguments, a running display of the entire server log is started.

Syntax

```
java.weblogic.Admin [Connection and User Credentials Arguments]
SERVERLOG [[starttime]|[endtime]]
```

Argument	Definition
starttime	Optional. Earliest time at which messages are to be displayed. If not specified, messages display starts, by default, when the `SERVERLOG` command is executed. The date format is *yyyy/mm/dd*. Time is indicated using a 24-hour clock. The start date and time are entered inside quotation marks, in the following format: "*yyyy/mm/dd hh:mm*"
endtime	Optional. Latest time at which messages are to be displayed. If not specified, the default is the time at which the `SERVERLOG` command is executed. The date format is *yyyy/mm/dd*. Time is indicated using a 24-hour clock. The end date and time are entered inside quotation marks, in the following format: "*yyyy/mm/dd hh:mm*"

Example

In the following example, a request is made for a running display of the log for the server listening on port 7001 on machine `localhost`.

```
java weblogic.Admin -url localhost:7001 -username adminuser
-password gumby1234 SERVERLOG "2001/12/01 14:00" "2001/12/01 16:00"
```

The request specifies that the running display should begin at 2:00 p.m. on December 1, 2001, and end at 4:00 p.m. on December 1, 2001.

SHUTDOWN

Shuts down the specified WebLogic Server.

When you issue this command, the server invokes any shutdown classes that you have configured. It then notifies all applications and subsystems to stop receiving new requests from external clients and to complete all current work. You can shut down a server gracefully only from the `RUNNING` or `STANDBY` states.

In release 6.x, this command included an option to specify a number of seconds to wait before starting the shutdown process. This option is now deprecated. To support this deprecated option, this command must assume that any numerical value that you supply in the field immediately after the `SHUTDOWN` command is intended to express seconds. You cannot use this command to gracefully shut down a server whose name is made up entirely of numbers. Instead, you must use the Administration Console. For information, refer to Shutting Down a Server in the Administration Console Online Help.

Instead of specifying a delay in the `SHUTDOWN` command, you can now set attributes to control how a server shuts down. For more information, refer to Timeout Period for LifeCycle Operations in the *WebLogic Administration Guide* and Setting the Timeout Period for LifeCycle Operations in the Administration Console Online Help.

Syntax

```
java weblogic.Admin [Connection and User Credentials Arguments]
SHUTDOWN [targetserver]

(Deprecated) java weblogic.Admin [-url URL] -username username
   [-password password] SHUTDOWN
   [seconds ]  ["lockMessage"] ]
```

Argument	Definition
targetserver	The name of the server to shut down. If you do not specify a value, the command shuts down the server that you specified in the `-url` argument.
seconds	(Deprecated) Optional. Number of seconds allowed to elapse between the invoking of this command and the shutdown of the server.

Argument	Definition
"*lockMessage*"	(Deprecated) Optional. Message, in double quotes, to be supplied in the message that is sent if a user tries to log in while the WebLogic Server is locked.

Example

In the following example, a user with the `adminuser` username and password `gumby1234` shuts down a server named MyServer via the Administration Server:

```
java weblogic.Admin -url MyAdminServer:7001 -username adminuser
-password gumby1234 SHUTDOWN MyServer
```

START

Starts a remote Managed Server using Node Manager.

Starts a remote Managed Server using Node Manager.

This command requires the following environment:

- The domain's Administration Server must be running.
- The Node Manager must be running on the Managed Server's host machine.
- The Managed Server's startup items and Node Manager settings must be set up as described in Managing Server Availability with Node Manager.

For information on performing this task from the Administration Console, refer to Starting a Server in the Administration Console Online Help.

Note: In the Administration Console, the Servers→General tab includes a Startup Mode field that you use to specify the state in which a server starts. However, this setting only applies if you start a server from the local host using the `weblogic.Server` command. The Node Manager, and therefore the `weblogic.Admin START` command, does not use the value that you specify. For example, even if you specify `STANDBY` as the value for the Startup Mode, if you issue the `weblogic.Admin START` command, the server will start in the `RUNNING` state.

Syntax

```
java weblogic.Admin [Connection and User Credentials Arguments]
   START targetserver
```

Argument	Definition
`-url` *`URL`*	Must specify the listen address of the domain's Administration Server. The default is `localhost:7001`.
`targetserver`	The name of the Managed Server to start in a `RUNNING` state.

Example

In the following example, a user with the `adminuser` username and password `gumby1234` attempts to start a server named MyServer via the Administration Server:

```
java weblogic.Admin -url myAdminServer:7001 -username adminuser
-password gumby1234 START MyServer
```

STARTINSTANDBY

Starts a remote Managed Server using Node Manager and places it in a `STANDBY` state. In this state, a server is not accessible to requests from external clients.

This command requires the following environment:

- The domain's Administration Server must be running.
- The Node Manager must be running on the Managed Server's host machine.
- The Managed Server's startup items and Node Manager settings must be set up as described in Managing Server Availability with Node Manager.
- The domain must be configured to use a domain-wide administration port as described in Configuring a Domain-Wide Administration Port.

Note: In the Administration Console, the Servers→General tab includes a Startup Mode field that you use to specify the state in which a server starts. However, this setting only applies if you start a server from the local host using the `weblogic.Server` command. The Node Manager, and therefore the `weblogic.Admin STARTINSTANDBY` command, does not use the value that you specify. For example, even if you specify `RUNNING` as the value for the Startup Mode, if you issue the `weblogic.Admin STARTINSTANDBY` command, the server will start in the `STANDBY` state.

For information on performing this task from the Administration Console, refer to Starting a Server in the STANDBY State in the Administration Console Online Help. For more information about server states, refer to The Server Lifecycle.

Syntax

```
java weblogic.Admin [Connection and User Credentials Arguments]
STARTINSTANDBY [targetserver]
```

Argument	Definition
-url	Must specify the domain's Administration Server. The default is `localhost:7001`.

Argument	Definition
`targetserver`	Optional. The name of the WebLogic Server to start in the `STANDBY` state. If you do not specify a value, the command starts the server that you specified in the `-url` argument.

Example

In the following example, a user with the `adminuser` username and password `gumby1234` attempts to start a server named MyServer via the Administration Server:

```
java weblogic.Admin -url myAdminServer:7001 -username adminuser
-password gumby1234 STARTINSTANDBY MyServer
```

STOREUSERCONFIG

Creates a user-configuration file and an associated key file. The user-configuration file contains an encrypted username and password. The key file contains a secret key that is used to encrypt and decrypt the username and password.

When you use other `weblogic.Admin` or `weblogic.Deployer` commands, you can specify the user-configuration file and key file instead of entering an unencrypted username and password on the command line or including unencrypted credentials in scripts. See "Summary of User Credentials Arguments" on page B-5.

Only the key file that originally encrypted the username and password can decyrpt the values. If you lose the key file, you must create a new user-configuration and key file pair.

Caution: You must ensure that only authorized users can access the key file. Any user who accesses a valid user-configuration and key file pair gains the privileges of the encrypted username. To secure access to the key file, you can store the key file in a directory that provides read and write access only to authorized users, such as WebLogic Server administrators. Alternatively, you can write the key file to a removable medium, such as a floppy or CD, and lock the medium in a drawer when it is not being used.

Unlike other `weblogic.Admin` commands, the `STOREUSERCONFIG` command does not connect to a WebLogic Server instance. The data encryption and file creation are accomplished by the JVM in which the `STOREUSERCONFIG` command runs. Because it does not connect to a WebLogic Server instance, the command cannot verify that the username and password are valid WebLogic Server credentials.

Syntax

```
java weblogic.Admin
  -username username [-password password]
  [ -userconfigfile config-file ] [ -userkeyfile keyfile ]
  STOREUSERCONFIG
```

Argument	Definition
`-userconfigfile` *`config-file`*	Specifies a file pathname at which the `STOREUSERCONFIG` command creates a user-configuration file. The pathname can be absolute or relative to the directory from which you enter the command. If a file already exists at the specified pathname, the command overwrites the file with a new file that contains the newly encrypted username and password. If you do not specify this option, `STOREUSERCONFIG` does the following: ■ To determine the directory in which to create the user-configuration file, it uses the JVM user-home directory. The default value varies depending on the SDK and type of operating system. See "Configuring the Default Path Name" on page B-32. ■ To determine the file name, it prepends your operating-system username to the string `-WebLogicConfig.properties`. For example, *`username`*`-WebLogicConfig.properties`. You can use Java options to specify a different username. See "Configuring the Default Path Name" on page B-32.
`-userkeyfile` *`keyfile`*	Specifies a file pathname at which the `STOREUSERCONFIG` command creates a key file. The pathname can be absolute or relative to the directory from which you enter the command. If a file already exists at the specified pathname, `STOREUSERCONFIG` uses the existing key file to encrypt the new user-configuration file. If you do not specify this option, `STOREUSERCONFIG` does the following: ■ To determine the directory in which to create the key file, it uses the JVM user-home directory. The default value varies depending on the SDK and type of operating system. See "Configuring the Default Path Name" on page B-32. ■ To determine the file name, it prepends your operating-system username to the string `-WebLogicKey.properties`. For example, *`username`*`-WebLogicKey.properties`. You can use Java options to specify a different username. See "Configuring the Default Path Name" on page B-32.

Argument	Definition (Continued)
`-username` *`username`* `[-password` *`password`*`]`	Specifies the username and password to encrypt. The `STOREUSERCONFIG` command does **not** verify that the username and password are valid WebLogic Server user credentials. If you omit the `-password` *`password`* argument, `STOREUSERCONFIG` prompts you to enter a password.

Configuring the Default Path Name

If you do not specify the location in which to create and use a user-configuration file and key file, the `weblogic.Admin` and `weblogic.Deployer` utilities supply the following default values:

- *`user-home-directory`*`\`*`username`*`-WebLogicConfig.properties`
- *`user-home-directory`*`\`*`username`*`-WebLogicKey.properties`

Where *`user-home-directory`* is the home directory of the operating-system user account as determined by the JVM, and *`username`* is your operating-system username.

The value of the home directory varies depending on the SDK and type of operating system. For example, on UNIX, the home directory is usually "~*`username`*." On Windows, the home directory is usually "`C:\Documents and Settings\`*`username`*".

You can use the following Java options to specify values for *`user-home-directory`* and *`username`*:

- `-Duser.home=`*`pathname`* specifies the value of *`user-home-directory`*
- `-Duser.name=`*`usernanme`* specifies the value of *`username`*.

For example, the following command configures the user-home directory to be `c:\myHome` and the user name to be `wlAdmin`. The command will search for the following user-configuration file and user key file:

```
c:\myHome\wlAdmin-WebLogicConfig.properties
c:\myHome\wlAdmin-WebLogicKey.properties
```

```
java -Duser.home=c:\myHome -Duser.name=wlAdmin
weblogic.Admin COMMAND
```

Creating User-Configuration and Key Files

To create user-configuration and key files:

1. Use the `-username` *`username`* and `-password` *`password`* arguments to specify the username and password to be encrypted.

2. Specify the name and location of the user-configuration and key files by doing one of the following:

   - Use the `-userconfigfile` *`config-file`* and `-userkeyfile` *`key-file`* arguments:
     ```
     java weblogic.Admin -username username -password password
      -userconfigfile config-file -userkeyfile key-file
     STOREUSERCONFIG
     ```
   - Use the default behavior to create files named *`user-home-directory`*`\`*`username`*`-WebLogicConfig.properties` and *`user-home-directory`*`\`*`username`*`-WebLogicKey.properties`:
     ```
     java weblogic.Admin -username username -password password
     STOREUSERCONFIG
     ```
   - Use the `-Duser.home=`*`directory`* and `-Duser.name=`*`username`* Java options to create files named *`directory`*`\`*`username`*`-WebLogicConfig.properties` and *`directory`*`\`*`username`*`-WebLogicKey.properties`:
     ```
     java -Duser.home=directory -Duser.name=username
     weblogic.Admin -username username -password password
     STOREUSERCONFIG
     ```

You can change the name and location of a user-configuration file or a key file after you create them, as long as you use the two files as a pair.

Using a Single Key File for Multiple User-Configuration Files

To use one key file to encrypt multiple user-configuration files:

1. Create an initial user-configuration file and key file pair.

   For example, enter the following command:

   ```
   java weblogic.Admin -username username -password password
   -userconfigfile c:\AdminConfig -userkeyfile e:\myKeyFile
   STOREUSERCONFIG
   ```

2. When you create an additional user-configuration file, specify the existing key file.

   For example, enter the following command:

   ```
   java weblogic.Admin -username username -password password
   -userconfigfile c:\anotherConfigFile -userkeyfile e:\myKeyFile
   STOREUSERCONFIG
   ```

Examples

In the following example, a user who is logged in to a UNIX operating system as `joe` encrypts the username `wlAdmin` and password `wlPass`:

```
java weblogic.Admin -username wlAdmin -password wlPass
STOREUSERCONFIG
```

The command determines whether a key file named `~joe/joe-WebLogicKey.properties` exists. If such a file does not exist, it prompts the user to select `y` to confirm creating a key file. If the command succeeds, it creates two files:
`~joe\joe-WebLogicConfig.properties`
`~joe\joe-WebLogicKey.properties`

The file `joe-WebLogicConfig.properties` contains an encrypted version of the strings `wlAdmin` and `wlPass`. Any command that uses the `~joe\joe-WebLogicConfig.properties` file must specify the `~joe\joe-WebLogicKey.properties` key file.

In the following example, the user `joe` is a System Administrator who wants to create a user-configuration file for an operating-system account named `pat`. For the sake of convenience, `joe` wants to create the user-configuration file in `pat`’s home directory, which will simplify the syntax of the `weblogic.Admin` commands that `pat` invokes. For added security, only one key file exists at `joe`’s organization, and it is located on a removable hard drive.

To create a user configuration file in `pat`’s home directory that is encrypted and decrypted by a key file name `e:\myKeyFile`:

```
java -Duser.name=pat -Duser.home="C:\Documents and Settings\pat"
weblogic.Admin -username wlOperatorPat -password wlOperator1
-userkeyfile e:\myKeyFile
STOREUSERCONFIG
```

A user who logs in to `pat`'s account can use the following syntax to invoke `weblogic.Admin` commands:

```
java weblogic.Admin -userkeyfile e:\myKeyFile COMMAND
```

For information on using user-configuration and key files, see "Summary of User Credentials Arguments" on page B-5.

THREAD_DUMP

Prints a snapshot of the WebLogic Server threads that are currently running for a specific server instance. The server instance prints the snapshot to its standard out.

Note: The `THREAD_DUMP` command is supported only on Sun JVM and JRockit.

Syntax

```
java weblogic.Admin [Connection and User Credentials Arguments]
THREAD_DUMP
```

Example

The following example causes a server instance that is running on a host named ManagedHost to print a thread dump to standard out:

```
java weblogic.Admin -url ManagedHost:8001 -username weblogic
-password weblogic THREAD_DUMP
```

If the command succeeds, the command itself returns the following:

```
Thread Dump is available in the command window that is running the
server.
```

The server instance prints a thread dump to its standard out, which, by default, is the shell (command prompt) within which the server instance is running.

UNLOCK

(Deprecated) Unlocks the specified WebLogic Server after a `LOCK` operation.

This command is deprecated because the `LOCK` command is deprecated. Instead of `LOCK` and `UNLOCK`, use `STARTINSTANDY` and `RESUME`. For more information, refer to "RESUME" on page B-22.

Syntax

```
java weblogic.Admin [Connection and User Credentials Arguments]
UNLOCK
```

Argument	Definition
username	Required. A valid administrative username must be supplied to use this command.
password	Required. A valid administrative password must be supplied to use this command.

Example

In the following example, an administrator named `adminuser` with a password of `gumby1234` requests the unlocking of the WebLogic Server listening on port 7001 on machine `localhost`:

```
java weblogic.Admin -url localhost:7001 -username adminuser
   -password gumby1234 UNLOCK
```

VERSION

Displays the version of the WebLogic Server software that is running on the machine specified by the value of *URL*.

Syntax

```
java weblogic.Admin [Connection and User Credentials Arguments]
VERSION
```

Example

In the following example, a user requests the version of the WebLogic Server running on port 7001 on machine `localhost`:

```
java weblogic.Admin -url localhost:7001 -username
installadministrator
   -password installadministrator VERSION
```

Note: In this example, the default value of both the *`username`* and `password` arguments, `installadministrator`, is used.

WebLogic Server Connection Pools Administration Command Reference

Table B-2 presents an overview of WebLogic Server administration commands for connection pools. The following sections describe command syntax and arguments, and provide an example for each command.

For additional information about connection pools see *Programming WebLogic JDBC* at `http://e-docs.bea.com/wls/docs70/jdbc/index.html` and Managing JDBC Connectivity in the *Administration Guide* at `http://e-docs.bea.com/wls/docs70/adminguide/jdbc.html`.

Table B-2 WebLogic Server Administration Commands Overview—Connection Pools

Task	Command	Description
Create a Dynamic Connection Pool	CREATE_POOL	Allows creation of connection pool while WebLogic Server is running. Note that dynamically created connection pools cannot be used with DataSources or TxDataSources. See "CREATE_POOL" on page B-41
Destroy a Connection Pool	DESTROY_POOL	Connections are closed and removed from the pool and the pool dies when it has no remaining connections. See "DESTROY_POOL" on page B-44.
Disable a Connection Pool	DISABLE_POOL	You can temporarily disable a connection pool, preventing any clients from obtaining a connection from the pool. See "DISABLE_POOL" on page B-45.
Enable a Connection Pool	ENABLE_POOL	When a pool is enabled after it has been disabled, the JDBC connection states for each in-use connection are exactly as they were when the connection pool was disabled; clients can continue JDBC operations exactly where they left off. See "ENABLE_POOL" on page B-46.

Table B-2 WebLogic Server Administration Commands Overview—Connection Pools

Task	Command	Description
Determine if a Connection Pool Exists	EXISTS_POOL	Tests whether a connection pool with a specified name exists in the WebLogic Server. You can use this command to determine whether a dynamic connection pool has already been created or to ensure that you select a unique name for a dynamic connection pool you want to create. See "EXISTS_POOL" on page B-47.
Resets a Connection Pool	RESET_POOL	Closes and reopens all allocated connections in a connection pool. This may be necessary after the DBMS has been restarted, for example. Often when one connection in a connection pool has failed, all of the connections in the pool are bad. See "RESET_POOL" on page B-48.

CREATE_POOL

Allows creation of connection pool while WebLogic Server is running. For more information, see “Creating a Connection Pool Dynamically” in *Programming WebLogic JDBC* at `http://e-docs.bea.com/wls/docs70/jdbc/programming.html#programming004`.

Syntax

```
java weblogic.Admin [Connection and User Credentials Arguments]
CREATE_POOL poolName  aclName=aclX,
   props=myProps,initialCapacity=1,maxCapacity=1,
   capacityIncrement=1,allowShrinking=true,shrinkPeriodMins=15,
   driver=myDriver,url=myURL
```

Argument	Definition
`poolName`	Required. Unique name of pool.
`aclName`	Required. Identifies the different access lists within `fileRealm.properties` in the server config directory. Paired name must be dynaPool.
`props`	Database connection properties; typically in the format “database login name; database password; server network id”.
initialCapacity	Initial number of connections in a pool. If this property is defined and a positive number > 0, WebLogic Server creates these connections at boot time. Default is 1; cannot exceed maxCapacity.
maxCapacity	Maximum number of connections allowed in the pool. Default is 1; if defined, maxCapacity should be =>1.
capacityIncrement	Number of connections that can be added at one time. Default = 1.
allowShrinking	Indicates whether or not the pool can shrink when connections are detected to not be in use. Default = true.
shrinkPeriodMins	Required. Interval between shrinking. Units in minutes. Minimum = 1.If `allowShrinking = True`, then default = 15 minutes.

Argument	Definition
driver	Required. Name of JDBC driver. Only local (non-XA) drivers can participate.
url	Required. URL of the JDBC driver.
testConnsOnReserve	Indicates reserved test connections. Default = False.
testConnsOnRelease	Indicates test connections when they are released. Default = False.
testTableName	Database table used when testing connections; must be present for tests to succeed. Required if either testConnOnReserve or testConOnRelease are defined.
refreshPeriod	Sets the connection refresh interval. Every unused connection will be tested using TestTableName. Connections that do not pass the test will be closed and reopened in an attempt to reestablish a valid physical database connection. If TestTableName is not set then the test will not be performed.
loginDelaySecs	The number of seconds to delay before creating each physical database connection. This delay takes place both during initial pool creation and during the lifetime of the pool whenever a physical database connection is created. Some database servers cannot handle multiple requests for connections in rapid succession. This property allows you to build in a small delay to let the database server catch up. This delay takes place both during initial pool creation and during the lifetime of the pool whenever a physical database connection is created.

Example

In the following example, a user with the name `weblogic` and the password `weblogic` runs the `CREATE_POOL` command to create a dynamic connection pool:

```
java weblogic.Admin -url localhost:7001 -username weblogic
   -password weblogic CREATE_POOL MyPool
```

```
java weblogic.Admin -url t3://forest:7901 -username weblogic
   -password weblogic CREATE_POOL dynapool6 "aclName=someAcl,
   allowShrinking=true,shrinkPeriodMins=10,
   url=jdbc:weblogic:oracle,driver=weblogic.jdbc.oci.Driver,
initialCapacity=2,maxCapacity=8,
props=user=SCOTT;password=tiger;server=bay816"
```

DESTROY_POOL

Connections are closed and removed from the pool and the pool dies when it has no remaining connections.

Syntax

```
java weblogic.Admin [Connection and User Credentials Arguments]
DESTROY_POOL poolName [true|false]
```

Argument	Definition
poolName	Required. Unique name of pool.
false (soft shutdown)	Soft shutdown waits for connections to be returned to the pool before closing them.
true (default—hard shutdown)	Hard shutdown kills all connections immediately. Clients using connections from the pool get exceptions if they attempt to use a connection after a hard shutdown.

Example

In the following example, a user with the name `adminuser` and the password `gumby1234` runs the `DESTROY_POOL` command temporarily freeze the active pool connections:

```
java weblogic.Admin -url localhost:7001 -username adminuser
   -password gumby1234 DESTROY_POOL MyPool false
```

DISABLE_POOL

You can temporarily disable a connection pool, preventing any clients from obtaining a connection from the pool.

You have to options for disabling a pool. 1) Freezing the connections in a pool that you later plan to enable, and 2) destroy the connections.

Syntax

```
java weblogic.Admin [Connection and User Credentials Arguments]
DISABLE_POOL poolName [true|false]
```

Argument	Definition
poolName	Name of the connection pool
false (disables and **suspends**)	Disables the connection pool, and suspends clients that currently have a connection. Attempts to communicate with the database server throw an exception. Clients can, however, close their connections while the connection pool is disabled; the connections are then returned to the pool and cannot be reserved by another client until the pool is enabled.
true (default—disables and **destroys**)	Disables the connection pool, and destroys the client's JDBC connection to the pool. Any transaction on the connection is rolled back and the connection is returned to the connection pool.

Example

In the following example, a user with the name `adminuser` and the password `gumby1234` runs the `DISABLE_POOL` command to freeze a connection that is to be enabled later:

```
java weblogic.Admin -url localhost:7001 -username adminuser
   -password gumby1234 DISABLE_POOL MyPool false
```

ENABLE_POOL

When a pool is enabled, the JDBC connection states for each in-use connection are exactly as they were when the connection pool was disabled; clients can continue JDBC operations exactly where they left off.

Syntax

```
java weblogic.Admin [Connection and User Credentials Arguments]
ENABLE_POOL poolName
```

Argument	Definition
poolName	Name of the connection pool.

Example

In the following example, a user with the name `adminuser` and the password `gumby1234` runs the `ENABLE_POOL` command to reestablish connections that have been disabled (frozen):

```
java weblogic.Admin -url localhost:7001 -username adminuser
   -password gumby1234 ENABLE_POOL MyPool
```

EXISTS_POOL

Tests whether a connection pool with a specified name exists in the WebLogic Server. You can use this method to determine whether a dynamic connection pool has already been created or to ensure that you select a unique name for a dynamic connection pool you want to create.

Syntax

```
java weblogic.Admin [Connection and User Credentials Arguments]
EXISTS_POOL poolName
```

Argument	Definition
poolName	Name of connection pool.

Example

In the following example, a user with the name `adminuser` and the password `gumby1234` runs the `EXISTS_POOL` command to determine wether or not a pool with a specific name exists:

```
java weblogic.Admin -url localhost:7001 -username adminuser
   -password gumby1234 EXISTS_POOL MyPool
```

RESET_POOL

This command resets the connections in a registered connection pool.

This is a privileged command. You must supply the username and password for the WebLogic Server administrative user to use this command. You must know the name of the connection pool, which is an entry in the `config.xml` file.

Syntax

```
 java weblogic.Admin [Connection and User Credentials Arguments]
RESET_POOL poolName
```

Argument	Definition
URL	The URL of the WebLogic Server host and port number of the TCP port at which WebLogic is listening for client requests; use `"t3://host:port."`
username	Required. A valid administrative username must be supplied to use this command.
password	Required. A valid administrative password must be supplied to use this command.
poolName	Name of a connection pool as it is registered in the WebLogic Server's config.xml file.

Example

This command refreshes the connection pool registered as "demoPool" for the WebLogic Server instance listening on port 7001 of the host xyz.com.

```
 java weblogic.Admin -url t3://xyz.com:7001 -username system
   -password gumby RESET_POOL demoPool
```

MBean Management Command Reference

Table B-3 presents an overview of the MBean management commands. The following sections describe command syntax and arguments, and provide an example for each command.

Table B-3 MBean Management Command Overview

Task	Command(s)	Description
Create Administration MBeans	CREATE	Creates an Administration MBean. Returns `OK` to `stdout` when successful. This command cannot be used for Runtime MBeans and we recommend that you do not use it to create Local Configuration MBeans. See "CREATE" on page B-52.
Delete MBeans	DELETE	Deletes an MBean. Returns `OK` in `stdout` when successful. See "DELETE" on page B-54.
View MBean properties (attributes)	GET	Displays properties of MBeans. See "GET" on page B-56.
Invoke MBean operations	INVOKE	Invokes management operations that an MBean exposes for its underlying resource. See "INVOKE" on page B-59.
Set property values for Administration MBeans or Local Configuration MBeans	SET	Sets the specified property values for the named MBean. Returns `OK` on `stdout` when successful. This command cannot be used for Runtime MBeans. See "SET" on page B-61.

Specifying MBean Types

All of the MBean management commands can accept a `-type` argument, which causes the command to operate on all MBeans that are an instance of a type that you specify. An MBean's **type** refers to the interface class of which the MBean is an instance. All WebLogic Server MBeans are an instance of one of the interface classes defined in the `weblogic.management.configuration` or `weblogic.management.runtime` packages. For configuration MBeans, type also refers to whether an instance is an Administration MBean or a Local Configuration MBean. For a complete list of all WebLogic Server MBean interface classes, refer to the WebLogic Server Javadoc for the `weblogic.management.configuration` or `weblogic.management.runtime` packages.

To determine the value that you provide for the `-type` argument, do the following:

1. Find the MBean's interface class and remove the `MBean` suffix from the class name. For an MBean that is an instance of the `weblogic.management.runtime.JDBCConnectionPoolRuntimeMBean`, use `JDBCConnectionPoolRuntime`.
2. For a Local Configuration MBean, append `Config` to the name. For example, for a Local Configuration MBean that is an instance of the `weblogic.management.configuration.JDBCConnectionPoolMBean` interface class, use `JDBCConnectionPoolConfig`. For the corresponding Administration MBean instance, use `JDBCConnectionPool`.

Specifying Servers

All of the MBean management commands include a `-url` argument that you use to specify the WebLogic Server instance that hosts the MBean.

To work with Administration MBeans, you must use the `-url` argument to specify the Administration Server. To work with Local Configuration MBeans or Runtime MBeans, you must use the `-url` argument to specify the WebLogic Server instance that hosts the MBeans.

If **all** of the following conditions are true, you can use `weblogic.Admin` to access any MBean in the entire domain:

- Your Administration Server listens on port 7001
- You issue the `weblogic.Admin` command from the Administration Server
- You omit the `-url` argument

For example, if all of the above conditions are true, then you can determine the state of all servers in the domain by issuing the following command:

```
java weblogic.Admin -username weblogic -password weblogic GET -type
ServerRuntime -property State
```

CREATE

Creates an instance of a WebLogic Server Administration MBean. Returns `OK` to `stdout` when successful. This command cannot be used for Runtime MBeans and we recommend that you do not use it to create Local Configuration MBeans.

When you use this command to create an Administration MBean instance, WebLogic Server populates the MBean with default values and saves the MBean's configuration in the domain's `config.xml` file. WebLogic Server does not create the corresponding Local Configuration MBean replica until you restart the server instance that hosts the underlying managed resource. For example, if you create a `JDBCConnectionPool` Administration MBean to manage a JDBC connection pool on a Managed Server named `MyServer`, you must restart `MyServer` so that it can create its local replica of the `JDBCConnectionPool` Administration MBean that you created. For more information on MBean replication and the lifecycle of MBeans, refer to "MBeans for Configuring Managed Resources" in the *Programming WebLogic Management Services* with JMX guide.

Syntax

```
java weblogic.Admin [Connection and User Credentials Arguments]
CREATE -name name -type mbean_type
   [-domain domain_name]

java weblogic.Admin [-url URL] -username username
   [-password password] CREATE -mbean mbean_name
```

Argument	Definition
URL	The WebLogic Server instance that is the target of the command. For more information, refer to "Specifying Servers" on page B-50.
name	The name you choose for the MBean that you are creating.
mbean_type	The type of MBean that you are creating. For more information, refer to "Specifying MBean Types" on page B-49.
mbean_name	Fully qualified object name of an MBean in the `WebLogicObjectName` format. For example: `"`*`domain`*`:Type=`*`type`*`,Name=`*`name`*`"` For more information, refer to the Javadoc for `WebLogicObjectName`.

Argument	Definition
domain_name	Name of the domain in which you want to create the MBean instance. If *domain_name* is not specified, the command assumes the domain to which the target server belongs.

Example

The following example uses the `-name` and `-type` arguments to create a `JDBCConnectionPool` Administration MBean named `MyPool` on an Administration Server:

```
java weblogic.Admin -url AdminHost:7001 -username weblogic
   -password weblogic CREATE -name MyPool -type JDBCConnectionPool
```

If the command succeeds, it prints the following to standard out:

```
Ok
```

The following example uses the `-mbean` argument and `WebLogicObjectName` conventions to create a `JDBCConnectionPool` Administration MBean named `MyPool` on an Administration Server:

```
java weblogic.Admin -url AdminHost:7001 -username weblogic
   -password weblogic
    CREATE -mbean “MyDomain:Type=JDBCConnectionPool,Name=MyPool”
```

DELETE

Deletes an MBean. If you delete an Administration MBean, WebLogic Server removes the corresponding entry from the domain's config.xml file. Returns `OK` in `stdout` when successful.

Note: When you delete an Administration MBean, a WebLogic Server instance does not delete the corresponding Configuration MBean until you restart the server instance.

Syntax

```
java weblogic.Admin [Connection and User Credentials Arguments]
 {-type mbean_type|-mbean mbean_name}
```

Arguments	Definition
URL	The WebLogic Server instance that is the target of the command. For more information, refer to "Specifying Servers" on page B-50.
mbean_type	Deletes all MBeans of the specified type. For more information, refer to "Specifying MBean Types" on page B-49.
mbean_name	Fully qualified object name of an MBean in the `WebLogicObjectName` format. For example: `"`*`domain`*`:Type=`*`type`*`,Name=`*`name`*`"` For more information, refer to the Javadoc for `WebLogicObjectName`.

Example

The following example deletes the `JDBCConnectionPool` Administration MBean named `MyPool`:

```
java weblogic.Admin -url AdminHost:7001 -username weblogic
 -password weblogic DELETE -mbean
 MyDomain:Name=MyPool,Type=JDBCConnectionPool
```

If the command succeeds, it prints the following to standard out:

```
Ok
```

The following example deletes the `JDBCConnectionPool` Local Configuration MBean named `MyPool` on a server instance named `MyServer`:

```
java weblogic.Admin -url ManagedHost:8001 -username weblogic
 -password weblogic DELETE -mbean
  MyDomain:Location=MyServer,Name=MyPool,
 Type=JDBCConnectionPoolConfig
```

The following example deletes all `JDBCConnectionPool` Local Configuration MBeans for all server instances in the domain:

```
java weblogic.Admin -adminurl AdminHost:7001 -username weblogic
   -password weblogic DELETE -type JDBCConnectionPoolConfig
```

The following example deletes all `JDBCConnectionPool` Local Configuration MBeans for a Managed Server that runs on a computer named `ManagedHost`:

```
java weblogic.Admin -url ManagedHost:8001 -username weblogic
 -password weblogic DELETE -type JDBCConnectionPoolConfig
```

GET

Displays MBean properties (attributes) and JMX object names (in the `WebLogicObjectName` format).

The output of the command is as follows:

```
{MBeanName object-name {property1 value} {property2 value}. . .}
{MBeanName object-name {property1 value} {property2 value} . . .}
. . .
```

Note that the properties and values are expressed as name-value pairs, each of which is returned within curly brackets. This format facilitates parsing of the output by a script.

If `-pretty` is specified, each property-value pair is displayed on a new line and curly brackets are not used to separate the pairs:

```
MBeanName: object-name
property1: value
property2: value
.
.
.
MBeanName: object-name
property1: value
abbribute2: value
```

Syntax

```
java weblogic.Admin [Connection and User Credentials Arguments]
 GET [-pretty] {-type mbean_type|-mbean mbean_name}
   [-property property1] [-property property2]...
```

Argument	Definition
URL	The WebLogic Server instance that is the target of the command. For more information, refer to "Specifying Servers" on page B-50.
mbean_type	Returns information for all MBeans of the specified type. For more information, refer to "Specifying MBean Types" on page B-49.

Argument	Definition
mbean_name	Fully qualified object name of an MBean in the `WebLogicObjectName` format: "*domain*:Type=*type*,Location:*location*,Name=*name*" For more information, refer to the Javadoc for `WebLogicObjectName`.
pretty	Places property-value pairs on separate lines.
property	The name of the MBean property (attribute) or properties to be listed. **Note:** If property is not specified using this argument, all properties are displayed.

Example

The following example displays all properties of the `JDBCConnectionPool` Administration MBean for a connection pool named `MyPool`. Note that the command must connect to the Administration Server to retrieve information from an Administration MBean:

```
java weblogic.Admin -url AdminHost:7001 -username weblogic
 -password weblogic GET -pretty -mbean
 MyDomain:Name=MyPool,Type=JDBCConnectionPool
```

If the command succeeds, it returns output similar to the following truncated example:

```
---------------------------
MBeanName: "MyDomain:Name=MyPool,Type=JDBCConnectionPool"
        ACLName:
        CachingDisabled: true
        CapacityIncrement: 1
        ConnLeakProfilingEnabled: false
        ConnectionCreationRetryFrequencySeconds: 0
        ConnectionReserveTimeoutSeconds: 10
...
```

The following example displays all instances of all `JDBCConnectionPoolRuntime` MBeans for all servers in the domain.

```
java weblogic.Admin -adminurl AdminHost:7001 -username weblogic
 -password weblogic GET -pretty -type JDBCConnectionPoolRuntime
```

The following example displays all instances of all `JDBCConnectionPoolRuntime` MBeans that have been deployed on the server instance that listens on `ManagedHost:8001`:

```
java weblogic.Admin -url ManagedHost:8001 -username weblogic
 -password weblogic GET -pretty -type JDBCConnectionPoolRuntime
```

INVOKE

Invokes a management operation for one or more MBeans. For WebLogic Server MBeans, you usually use this command to invoke operations other than the get*Attribute* and set*Attribute* that most WebLogic Server MBeans provide.

Syntax

```
java weblogic.Admin [Connection and User Credentials Arguments]
INVOKE {-type mbean_type|-mbean mbean_name} -method
   methodname [argument . . .]
```

Arguments	Definition
URL	The WebLogic Server instance that is the target of the command. For more information, refer to "Specifying Servers" on page B-50.
mbean_type	Invokes the operation on all MBeans of a specific type. For more information, refer to "Specifying MBean Types" on page B-49.
mbean_name	Fully qualified object name of an MBean, in the `WebLogicObjectName` format: "domain:Type=*type*,Location=*location*,Name=*name*" For more information refer to the Javadoc for `WebLogicObjectName`.
methodname	Name of the method to be invoked.
argument	Arguments to be passed to the method call. When the argument is a String array, the arguments must be passed in the following format: "*String1*;*String2*;. . . "

Example

The following example enables a JDBC connection pool by invoking the `enable` method of the `JDBCConnectionPoolRuntime` MBean:

```
java weblogic.Admin -url AdminHost:7001 -username weblogic
 -password weblogic INVOKE
 -mbean MyDomain:Location=MyServer,Name=MyPool,
  ServerRuntime=MyServer,Type=JDBCConnectionPoolRuntime
 -method enable
```

If the command succeeds, it returns the following:

```
{MBeanName="MyDomain:Location=MyServer,Name=MyPool,ServerRuntime=
MyServer,Type=JDBCConnectionPoolRuntime"}
```

```
Ok
```

The following example enables all JDBC connection pools in the domain by invoking the `enable` method of all the `JDBCConnectionPoolRuntime` MBeans:

```
java weblogic.Admin -adminurl AdminHost:7001 -username weblogic
 -password weblogic INVOKE
 -type JDBCConnectionPoolRuntime -method enable
```

SET

Sets the specified property (attribute) values for a configuration MBean. Returns `OK` on `stdout` when successful. This command cannot be used for runtime MBeans.

If you use this command for an Administration MBean, the new values are saved to the `config.xml` file or the security realm, depending on where the new values have been defined.

Syntax

```
java weblogic.Admin [Connection and User Credentials Arguments]
 SET {-type mbean_type|-mbean mbean_name}
   -property property1 property1_value
   [-property property2 property2_value] . . .
```

Argument	Definition
URL	The WebLogic Server instance that is the target of the command. For more information, refer to "Specifying Servers" on page B-50.
mbean_type	Sets the properties for all MBeans of a specific type. For more information, refer to "Specifying MBean Types" on page B-49.
mbean_name	Fully qualified object name of an MBean in the `WebLogicObjectName` format. For example: `"`*`domain`*`:Type=`*`type`*`,Name=`*`name`*`"` For more information, refer to the Javadoc for `WebLogicObjectName`.
property	The name of the property to be set.

Argument	Definition

property _value	The value to be set.
	• Some properties require you to specify the name of a WebLogic Server MBean. In this case, specify the fully qualified object name of an MBean in the `WebLogicObjectName` format. For example:
	`"`*`domain`*`:Type=`*`type`*`,Name=`*`name`*`"`
	For more information, refer to the Javadoc for `WebLogicObjectName`.
	• When the property value is an MBean array, separate each MBean object name by a semicolon and surround the entire property value list with quotes:
	`"`*`domain`*`:Name=`*`name`*`,Type=`*`type`*`;`*`domain`*`:Name=`*`name`*`,Type=`*`type`*`"`
	• When the property value is a String array, separate each string by a semicolon and surround the entire property value list with quotes:
	`"`*`String1`*`;`*`String2`*`;. . . "`
	• When the property value is a String or String array, you can set the value to null by using either of the following:
	`-property `*`property-name`*` ""`
	`-property `*`property-name`*
	For example, both `-property ListenAddress ""` and `-property ListenAddress` set the listen address to null.
	• If the property value contains spaces, surround the value with quotes:
	`"-Da=1 -Db=3"`
	For example:
	`SET -type ServerStart -property Arguments "-Da=1 -Db=3"`
	• When setting the properties for a JDBC Connection Pool, you must pass the arguments in the following format:
	`"user:`*`username`*`;password:`*`password`*`;server:`*`servername`*`"`

Example

The following example sets to `64` the `StdoutSeverityLevel` property of the local configuration instance of the `ServerMBean` for a server named `MyServer`:

```
java weblogic.Admin -url http://ManagedHost:8001
  -username weblogic -password weblogic
  SET -mbean
  MyDomain:Location=MyServer,Name=MyServer,Type=ServerConfig
 -property StdoutSeverityLevel 64
```

If the command succeeds, the server instance writes a log message similar to the following:

```
<Sep 16, 2001 12:11:27 PM EDT> <Info> <Logging> <000000> <Log
messages of every severity will be displayed in the shell console.>
```

The command prints `Ok` to standard out.

The following example sets to `64` the `StdoutSeverityLevel` property for all administration instances of `ServerMBean` in the current domain:

```
java weblogic.Admin -url http://AdminHost:7001
  -username weblogic -password weblogic
  SET -type Server -property StdoutSeverityLevel 64
```

Example: Targeting a JDBC Connection Pool

The following example targets a JDBC connection pool named `MyPool` to a server named `MS1`:

1. To determine the object name of the `MS1` WebLogic Server instance, enter the following command:

   ```
   java weblogic.Admin -url http://AdminHost:7001
     -username weblogic -password weblogic
      GET -type Server
   ```

   Because the `-url` argument specifies the Administration Server, the command returns a `Server` Administration MBean for all server instances in the domain. The output includes the object name for `ServerMBean` that represents `MS1`:

   ```
   MBeanName: "examples:Name=MS1,Type=Server"
           AcceptBacklog: 50
           AdministrationPort: 0
           AutoKillIfFailed: false
   .
   .
   .
   ```

2. To determine the object name of the JDBC Connection Pool that you want to target, enter the following command:

   ```
   java weblogic.Admin -url http://AdminHost:7001
     -username weblogic -password weblogic
      GET -type JDBCConnectionPool
   ```

The command returns a `JDBCConnectionPool` Administration MBean for all JDBC Connection Pools in the domain. The output includes the object name of the `MyPool` connection pool:

```
MBeanName: "examples:Name=MyPool,Type=JDBCConnectionPool"
        ACLName:
        CachingDisabled: true
        CapacityIncrement: 1

.
.
.
```

3. To target `MyPool` to `MS1`, enter the following command:

   ```
   java weblogic.Admin -url http://AdminHost:7001
     -username weblogic -password weblogic
     SET -mbean "examples:Name=MyPool,Type=JDBCConnectionPool"
     -property Targets "examples:Name=MS1,Type=Server"
   ```

   Because you must modify an Administration MBean (`JDBCConnectionPoolMBean`) to complete this task, the `-url` option must specify the domain's Administration Server. See "MBeans for Configuring Managed Resources" in *Programming WebLogic Management Services with JMX*.

Using weblogic.Admin Commands to Manage Users and Groups

In the WebLogic Security Service, an **Authentication provider** is the software component that proves the identity of users or system processes. An Authentication provider also remembers, transports, and makes that identity information available to various components of a system when needed. A security realm can use different types of Authentication providers to manage different sets of users and groups. See "Authentication Providers" in *Developing Security Providers for WebLogic Server.*

You can use the `weblogic.Admin` utility to invoke operations on the following types of Authentication providers:

- The WebLogic Authentication provider, `weblogic.management.security.authentication.AuthenticatorMBean.`

  By default, all security realms use this Authentication provider to manage users and groups.

- The LDAP Authentication providers that extend `weblogic.management.security.authentication.LDAPAuthenticatorMBean.`

- Custom Authentication providers that extend `weblogic.security.spi.AuthenticationProvider` and extend the optional Authentication SSPI MBeans. See "SSPI MBean Quick Reference" in *Developing Security Providers for WebLogic Server.*

The following sections describe basic tasks for managing users and groups with the `weblogic.Admin` utility:

- "Finding the Object Name for an AuthenticationProvider MBean" on page B-65
- "Creating a User" on page B-66
- "Adding a User to a Group" on page B-67
- "Verifying Whether a User is a Member of a Group" on page B-67
- "Listing Groups to Which a User Belongs" on page B-68
- "Limiting Group Membership Searching in an LDAP Server" on page B-68

For information about additional tasks that the `AuthenticationProvider` and the optional MBeans support, refer to the Javadoc for the `weblogic.management.security.authentication` package.

Finding the Object Name for an AuthenticationProvider MBean

To invoke operations on an Authentication provider, you must specify the object name of the provider's `AuthenticationProviderMBean`.

The object name of the WebLogic Authentication provider is:
`Security:Name=`*`realmName`*`DefaultAuthenticator`

where *`realmName`* is the name of a security realm. For example, `Security:Name=myrealmDefaultAuthenticator`.

The object name of the LDAP authentication providers is: `Security:Name=realmNameLDAPAuthenticator`

BEA recommends that you follow a similar convention when you create your own Authentication providers:

```
Security:Name=realmNameAuthenticatorName
```

If you use the Administration Console to add an Authentication provider to the realm, your `AuthenticationProviderMBean` name will follow the recommended naming convention.

Creating a User

To create a user, invoke the `UserEditorMBean.createUser` method, which is extended by the security realm's `AuthenticationProvider` MBean. See the Javadoc for the `createUser` method.

The method requires three input parameters:

```
username password user-description
```

Separate the parameters with a space. If any parameter contains a space, surround it with quotes.

The following example invokes `createUser` on the WebLogic Authentication provider:

```
java weblogic.Admin -adminurl localhost:8001 -username weblogic
-password weblogic invoke -mbean
Security:Name=myrealmDefaultAuthenticator
-method createUser my-user1 mypassword "my user"
```

If the command succeeds, it prints `OK` to standard out.

Adding a User to a Group

To add a user to a group, invoke the `GroupEditorMBean.addMemberToGroup` method, which is extended by the security realm's `AuthenticationProvider` MBean. See the Javadoc for the `addMemberToGroup` method.

The method requires two input parameters:

```
groupname username
```

The following example invokes `addMemberToGroup` on the WebLogic Authentication provider:

```
java weblogic.Admin -adminurl localhost:8001 -username weblogic
-password weblogic invoke -mbean
Security:Name=myrealmDefaultAuthenticator
-method addMemberToGroup Administrators my-user1
```

If the command succeeds, it prints `OK` to standard out.

Verifying Whether a User is a Member of a Group

To verify whether a user is a member of a group, invoke the `GroupEditorMBean.isMember` method, which is extended by the security realm's `AuthenticationProvider` MBean. See the Javadoc for the `isMember` method.

The method requires three input parameters:

```
groupname username boolean
```

where *`boolean`* specifies whether the command searches within child groups. If you specify `true`, the command returns `true` if the member belongs to the group that you specify or to any of the groups contained within that group.

The following example invokes `isMember` on the WebLogic Authentication provider:

```
java weblogic.Admin -adminurl localhost:8001 -username weblogic
-password weblogic invoke -mbean
Security:Name=myrealmDefaultAuthenticator
-method isMember Administrators weblogic true
```

If the user is a member of the group, the command prints `true` to standard out.

Listing Groups to Which a User Belongs

To see a list of groups that contain a user or a group, invoke the `MemberGroupListerMBean.listMemberGroups` method, which is extended by the security realm's `AuthenticationProvider` MBean. See the Javadoc for the `listMemberGroups` method.

The method requires one input parameter:

`memberUserOrGroupName`

where *`memberUserOrGroupName`* specifies the name of an existing user or a group.

The following example invokes `listMemberGroups` on the WebLogic Authentication Provider:

```
java weblogic.Admin -adminurl localhost:8001 -username weblogic
-password weblogic invoke -mbean
Security:Name=myrealmDefaultAuthenticator
-method listMemberGroups my-user1
```

The method returns a cursor, which refers to a list of names. The `weblogic.management.utils.NameLister.haveCurrent`, `getCurrentName`, and `advance` methods iterate through the returned list and retrieve the name to which the current cursor position refers. See the Javadoc for the `weblogic.management.utils.NameLister` interface.

Limiting Group Membership Searching in an LDAP Server

Unlimited, recursive searches for group membership can take up considerable time and produce a performance bottleneck. To control the depth of group membership searches, you can set the value of `GroupMembershipSearching` on the WebLogic Authentication provider, LDAP Authentication providers, or any other Authentication provider that implements the `AuthenticationProvider` interface.

To limit group searches, set the value of `GroupMembershipSearching` on the WebLogic Authentication provider or LDAP Authentication providers to `limited`. The `GroupMembershipSearching` attribute has the following valid values:

```
unlimited limited
```

where `limited` specifies whether the command searches within one or more levels of a nested group hierarchy. If you specify a limited search, the `MaxGroupMembershipSearchLevel` attribute must be specified. The default is an unlimited search.

The `MaxGroupMembershipSearchLevel` attribute specifies how many levels of group membership to search. Valid values are `0` and positive integers, where `0` specifies that the command searches only within direct group memberships. A positive integer specifies the number of levels to search.

For example, when searching for membership in Group A, `0` indicates that only direct members of Group A will be found. If Group B is a member of Group A, the members of Group B will not be found by this search. If the attribute is set to `1`, a search for membership in Group A will return direct members of Group A and any members of groups which are direct members of Group A. If Group B is a member of Group A, the members of Group B will also be found by this search. However, if Group C is a member of Group B, the members of Group C will not be found by this search.

See the Javadoc for `GroupMembershipSearching` and `MaxGroupMembershipSearchLevel`.

The following example gets the `GroupMembershipSearching` property on the WebLogic Authentication provider:

```
java weblogic.Admin -username xxxx -password yyyy -url
t3://localhost:7001 get -mbean
Security:Name=myrealmDefaultAuthenticator -pretty -property
GroupMembershipSearching -commotype
```

C Using Ant Tasks to Configure a WebLogic Server Domain

The following sections describe how to start and stop WebLogic Server instances and configure WebLogic Server domains using WebLogic Ant tasks:

Overview of Configuring and Starting Domains Using Ant Tasks

WebLogic Server provides a pair of Ant tasks to help you perform common configuration tasks in a development environment. The configuration tasks enable you to start and stop WebLogic Server instances as well as create and configure WebLogic Server domains.

Notes: The WebLogic Server Ant tasks are incompatible with Ant versions prior to 1.5. Also, if you use a version of Ant that is not included with WebLogic Server, you must specify the appropriate task definition in your `build.xml` file, as described in the sections that follow.

When combined with other WebLogic Ant tasks, you can create powerful build scripts for demonstrating or testing your application with custom domains. For example, a single Ant build script can:

- Create a new single-server domain and start the Administration Server using the `wlserver` Ant task.
- Configure the new domain with required application resources using the `wlconfig` Ant task.
- Deploy the application using the `wldeploy` Ant task.
- Automatically start a compiled client application to demonstrate or test product features.

The sections that follow describe how to use the configuration Ant tasks, `wlserver` and `wlconfig`. For more information about the `wldeploy` task, see wldeploy Ant Task in *Developing WebLogic Server Applications*.

Starting Servers and Creating Domains Using the wlserver Ant Task

What the wlserver Ant Task Does

The `wlserver` Ant task enables you to start, reboot, shutdown, or connect to a WebLogic Server instance. The server instance may already exist in a configured WebLogic Server domain, or you can create a new single-server domain for development by using the `generateconfig=true` attribute.

When you use the `wlserver` task in an Ant script, the task does not return control until the specified server is available and listening for connections. If you start up a server instance using wlserver, the server process automatically terminates after the Ant VM terminates. If you only connect to a currently-running server using the `wlserver` task, the server process keeps running after Ant completes.

Basic Steps for Using wlserver

To use the wlserver Ant task:

1. Set your environment.

   On Windows NT, execute the `setWLSEnv.cmd` command, located in the directory `WL_HOME\server\bin`, where `WL_HOME` is the top-level directory of your WebLogic Server installation.

   On UNIX, execute the `setWLSEnv.sh` command, located in the directory `WL_HOME/server/bin`, where `WL_HOME` is the top-level directory of your WebLogic Server installation.

   Note: The `wlserver` task is predefined in the version of Ant shipped with WebLogic Server. If you want to use the task with your own Ant installation, add the following task definition in your build file:

   ```
   <taskdef name="wlserver"
   classname="weblogic.ant.taskdefs.management.WLServer"/>
   ```

2. Add a call to the `wlserver` task in the build script to start, shutdown, restart, or connect to a server. See "wlserver Ant Task Reference" on page C-5 for information about `wlserver` attributes and default behavior.

3. Execute the Ant task or tasks specified in the `build.xml` file by typing `ant` in the staging directory, optionally passing the command a target argument:

   ```
   prompt> ant
   ```

   Use `ant -verbose` to obtain more detailed messages from the `wlserver` task.

Sample build.xml Files for wlserver

The following shows a minimal `wlserver` target that starts a server in the current directory using all default values:

```
  <target name="wlserver-default">
    <wlserver/>
  </target>
```

This target connects to an existing, running server using the indicated connection parameters and username/password combination:

```
  <target name="connect-server">
    <wlserver host="127.0.0.1" port="7001" username="weblogic"
password="weblogic" action="connect"/>
  </target>
```

This target starts a WebLogic Server instance configured in the `config` subdirectory:

```
  <target name="start-server">
    <wlserver dir="./config" host="127.0.0.1" port="7001"
action="start"/>
  </target>
```

This target creates a new single-server domain in an empty directory, and starts the domain's server instance:

```
  <target name="new-server">
    <delete dir="./tmp"/>
    <mkdir dir="./tmp"/>
    <wlserver dir="./tmp" host="127.0.0.1" port="7001"
    generateConfig="true" username="weblogic" password="weblogic"
action="start"/>
  </target>
```

wlserver Ant Task Reference

The following table describes the attributes of the `wlserver` Ant task.

Table 13-2 Attributes of the wlserver Ant Task

Attribute	Description	Data Type	Required?
policy	The path to the security policy file for the WebLogic Server domain. This attribute is used only for starting server instances.	File	No
dir	The path that holds the domain configuration (for example, `c:\bea\user_projects\mydomain`). By default, `wlserver` uses the current directory.	File	No
beahome	The path to the BEA home directory (for example, `c:\bea`).	File	No
weblogichome	The path to the WebLogic Server installation directory (for example, `c:\bea\weblogic700`).	File	No
servername	The name of the server to start, reboot, or connect to.	String	No
domainname	The name of the WebLogic Server domain in which the server is configured.	String	No
adminserverurl	The URL to access the Administration Server in the domain. This attribute is required if you are starting up a Managed Server in the domain.	String	Required for starting Managed Servers.
username	The username of an administrator account. If you omit both the `username` and `password` attributes, `wlserver` attempts to obtain the encrypted username and password values from the `boot.properties` file.	String	No
password	The password of an administrator account. If you omit both the `username` and `password` attributes, `wlserver` attempts to obtain the encrypted username and password values from the `boot.properties` file.	String	No
pkpassword	The private key password for decrypting the SSL private key file.	String	No

Table 13-2 Attributes of the wlserver Ant Task

Attribute	Description	Data Type	Required?
timeout	The maximum time, in seconds, that `wlserver` waits for a server to boot. This also specifies the maximum amount of time to wait when connecting to a running server.	long	No
productionmodeenabled	Specifies whether a server instance boots in development mode or in production mode.	boolean	No
host	The DNS name or IP address on which the server instance is listening.	String	No
port	The TCP port number on which the server instance is listening.	int	No
generateconfig	Specifies whether or not `wlserver` creates a new domain for the specified server. This attribute is false by default.	boolean	No
action	Specifies the action `wlserver` performs: `startup`, `shutdown`, `reboot`, or `connect`. The `shutdown` action can be used with the optional `forceshutdown` attribute perform a forced shutdown.	String	No
failonerror	This is a global attribute used by WebLogic Server Ant tasks. It specifies whether the task should fail if it encounters an error during the build. This attribute is set to true by default.	Boolean	No

forceshutdown	This optional attribute is used in conjunction with the `action="shutdown"` attribute to perform a forced shutdown. For example:
	
	`<wlserver`
	`  host="${wls.host}"`
	`  port="${port}"`
	`  username="${wls.username}"`
	`  password="${wls.password}"`
	`  action="shutdown"`
	`  forceshutdown="true"/>`
		Boolean	No

Configuring a WebLogic Server Domain Using the wlconfig Ant Task

What the wlconfig Ant Task Does

The `wlconfig` Ant task enables you to configure a WebLogic Server domain by creating, querying, or modifying configuration MBeans on a running Administration Server instance. Specifically, `wlconfig` enables you to:

- Create new MBeans, optionally storing the new MBean Object Names in Ant properties.
- Set attribute values on a named MBean available on the Administration Server.
- Create MBeans and set their attributes in one step by nesting set attribute commands within create MBean commands.
- Query MBeans, optionally storing the query results in an Ant property reference.
- Query MBeans and set attribute values on all matching results.
- Establish a parent/child relationship among MBeans by nesting create commands within other create commands.

Basic Steps for Using wlconfig

1. Set your environment in a command shell. See "Basic Steps for Using wlserver" on page C-3 for details.

   Note: The `wlconfig` task is predefined in the version of Ant shipped with WebLogic Server. If you want to use the task with your own Ant installation, add the following task definition in your build file:

   ```
   <taskdef name="wlconfig"
   classname="weblogic.ant.taskdefs.management.WLConfig"/>
   ```

2. `wlconfig` is commonly used in combination with `wlserver` to configure a new WebLogic Server domain created in the context of an Ant task. If you will be using `wlconfig` to configure such a domain, first use `wlserver` attributes to create a new domain and start the WebLogic Server instance.

3. Add an initial call to the `wlconfig` task to connect to the Administration Server for a domain. For example:

   ```
   <target name=”doconfig”>
      <wlconfig url="t3://localhost:7001" username="weblogic"
         password="weblogic">
   </target>
   ```

4. Add nested `create`, `delete`, `get`, `set`, and `query` elements to configure the domain.

5. Execute the Ant task or tasks specified in the `build.xml` file by typing `ant` in the staging directory, optionally passing the command a target argument:

   ```
   prompt> ant doconfig
   ```

   Use `ant -verbose` to obtain more detailed messages from the `wlconfig` task.

Sample build.xml Files for wlconfig

Complete Example

This example shows a single `build.xml` file that creates a new domain using `wlserver` and performs various domain configuration tasks with `wlconfig`.

The script starts by creating the new domain:

```
<target name="sample.config">
   mkdir dir="config"/>
   <wlserver username="a" password="a" servername="SampleServer"
      domainname="sample" dir="config" host="localhost"
port="7000"
      generateconfig="true"/>
```

The script then starts the `wlconfig` task by accessing the newly-created server:

```
   <wlconfig url="t3://localhost:7000" username="a" password="a">
```

Within the `wlconfig` task, the `query` element runs a query to obtain the Server MBean object name, and stores this MBean in the `${sampleserver}` Ant property:

```
 <query domain="sample" type="Server" name="SampleServer"
      property="sampleserver"/>
```

The script the uses a `create` element to create a new JDBC connection pool in the domain, storing the object name in the `${samplepool}` Ant property. Nested `set` elements in the `create` operation set attributes on the newly-created MBean. The new pool is target to the server using the `${sampleserver}` Ant property set in the query above:

```
   <create type="JDBCConnectionPool" name="SamplePool"
      property="samplepool">
      <set attribute="CapacityIncrement" value="1"/>
      <set attribute="DriverName"
         value="com.pointbase.jdbc.jdbcUniversalDriver"/>
      <set attribute="InitialCapacity" value="1"/>
      <set attribute="MaxCapacity" value="10"/>
      <set attribute="Password" value="samplename"/>
      <set attribute="Properties" value="user=samplepwd"/>
      <set attribute="RefreshMinutes" value="0"/>
      <set attribute="ShrinkPeriodMinutes" value="15"/>
      <set attribute="ShrinkingEnabled" value="true"/>
      <set attribute="TestConnectionsOnRelease" value="false"/>
      <set attribute="TestConnectionsOnReserve" value="false"/>
      <set attribute="URL"
         value="jdbc:pointbase:server://localhost/demo"/>
      <set attribute="Targets" value="${sampleserver}"/>
   </create>
```

Next, the script creates a JDBC TX DataSource using the JDBC connection pool created above:

```
   <create type="JDBCTxDataSource" name="Medical Records Tx
DataSource">
      <set attribute="JNDIName" value="SampleTxDataSource"/>
      <set attribute="PoolName" value="SamplePool"/>
      <set attribute="Targets" value="${sampleserver}"/>
   </create>
```

The script creates a new JMS connection factory using nested `set` elements:

```
   <create type="JMSConnectionFactory" name="Queue">
      <set attribute="JNDIName"
value="jms/QueueConnectionFactory"/>
      <set attribute="XAServerEnabled" value="true"/>
      <set attribute="Targets" value="${sampleserver}"/>
   </create>
```

A new JMS JDBC store is created using the `SamplePool`:

```
   <create type="JMSJDBCStore" name="SampleJDBCStore"
      property="samplejdbcstore">
      <set attribute="ConnectionPool" value="${samplepool}"/>
      <set attribute="PrefixName" value="Sample"/>
   </create>
```

When creating a new JMS server, the script uses a nested `create` element to create a JMS queue, which is the child of the JMS server:

```
   <create type="JMSServer" name="SampleJMSServer">
      <set attribute="Store" value="${samplejdbcstore}"/>
      <set attribute="Targets" value="${sampleserver}"/>
      <create type="JMSQueue" name="Registration Queue">
         <set attribute="JNDIName"
value="jms/REGISTRATION_MDB_QUEUE"/>
      </create>
   </create>
```

This script creates a new mail session and startup class:

```
   <create type="MailSession" name="Medical Records Mail Session">
      <set attribute="JNDIName" value="mail/SampleMailSession"/>
      <set attribute="Properties"
         value="mail.user=joe;mail.host=mail.mycompany.com"/>
      <set attribute="Targets" value="${sampleserver}"/>
   </create>

   <create type="StartupClass" name="StartBrowser">
      <set attribute="Arguments" value="port=${listenport}"/>
      <set attribute="ClassName"
         value="com.bea.sample.startup.StartBrowser"/>
      <set attribute="FailureIsFatal" value="false"/>
      <set attribute="Notes" value="Automatically starts a browser
on server boot."/>

      <set attribute="Targets" value="${sampleserver}"/>

    </create>
```

Finally, the script obtains the WebServer MBean and sets the log filename using a nested `set` element:

```
      <query domain="sample" type="WebServer" name="SampleServer">
         <set attribute="LogFileName" value="logs/access.log"/>
      </query>
```

```
   </wlconfig>
</target>
```

Query and Delete Example

The `query` element does not need to specify an MBean name when nested within a `query` element:

```
<target name="queryDelete">
   <wlconfig url="${adminurl}" username="${user}"
password="${pass}"
      failonerror="false">
      <query query="${wlsdomain}:Name=MyNewServer2,*"
         property="deleteQuery">
         <delete/>
      </query>
   </wlconfig>
</target>
```

Example of Setting Multiple Attribute Values

The `set` element allows you to set an attribute value to multiple object names stored in Ant properties. For example, the following target stores the object names of two servers in separate Ant properties, then uses those properties to assign both servers to the target attribute of a new JDBC Connection Pool:

```
<target name="multipleJDBCTargets">
   <wlconfig url="${adminurl}" username="${user}"
password="${pass}">
      <query domain="mydomain" type="Server" name="MyServer"
         property="myserver"/>
      <query domain="mydomain" type="Server" name="OtherServer"
         property="otherserver"/>
      <create type="JDBCConnectionPool" name="sqlpool"
property="sqlpool">
         <set attribute="CapacityIncrement" value="1"/>
[.....]
         <set attribute="Targets"
value="${myserver},${otherserver}"/>
      </create>
   </wlconfig>
</target>
```

Note: Separate multiple values with a comma (,) not a semi-colon (;) .

wlconfig Ant Task Reference

Main Attributes

The following table describes the main attributes of the `wlconfig` Ant task.

Table 13-3 Main Attributes of the wlconfig Ant Task

Attribute	Description	Data Type	Required?
url	The URL of the domain's Administration Server.	String	Yes
username	The username of an administrator account.	String	No
password	The password of an administrator account. **To avoid having the plain text password appear in the build file or in process utilities such as** `ps`, first store a valid username and encrypted password in a configuration file using the `weblogic.Admin STOREUSERCONFIG` command. Then omit both the `username` and `password` attributes in your Ant build file. When the attributes are omitted, `wlconfig` attempts to login using values obtained from the default configuration file. If you want to obtain a username and password from a non-default configuration file and key file, use the `userconfigfile` and `userkeyfile` attributes with `wlconfig`.	String	No
failonerror	This is a global attribute used by WebLogic Server Ant tasks. It specifies whether the task should fail if it encounters an error during the build. This attribute is set to true by default.	Boolean	No

Table 13-3 Main Attributes of the wlconfig Ant Task

Attribute	Description	Data Type	Required?
userconfigfile	Specifies the location of a user configuration file to use for obtaining the administrative username and password. Use this option, instead of the `username` and `password` attributes, in your build file when you do not want to have the plain text password shown in-line or in process-level utilities such as `ps`. Before specifying the `userconfigfile` attribute, you must first generate the file using the `weblogic.Admin STOREUSERCONFIG` command.	File	No
userkeyfile	Specifies the location of a user key file to use for encrypting and decrypting the username and password information stored in a user configuration file (the `userconfigfile` attribute). Before specifying the `userkeyfile` attribute, you must first generate the key file using the `weblogic.Admin STOREUSERCONFIG` command.	File	No

Nested Elements

`wlconfig` also has several elements that can be nested to specify configuration options:

- create
- delete
- set
- get
- query

create

The `create` element creates a new MBean in the WebLogic Server domain. The `wlconfig` task can have any number of `create` elements.

A `create` element can have any number of nested `set` elements, which set attributes on the newly-created MBean. A `create` element may also have additional, nested `create` elements that create child MBeans.

The `create` element has the following attributes.

Table 13-4 Attributes of the create Element

Attribute	Description	Data Type	Required?
name	The name of the new MBean object to create.	String	No (`wlconfig` supplies a default name if none is specified.)
type	The MBean type.	String	Yes
property	The name of an optional Ant property that holds the object name of the newly-created MBean.	String	No

delete

The `delete` element removes an existing MBean from the WebLogic Server domain. `delete` takes a single attribute:

Table 13-5 Attribute of the delete Element

Attribute	Description	Data Type	Required?
mbean	The object name of the MBean to delete.	String	Required when the `delete` element is a direct child of the `wlconfig` task. Not required when nested within a `query` element.

set

The `set` element sets MBean attributes on a named MBean, a newly-created MBean, or on MBeans retrieved as part of a query. You can include the `set` element as a direct child of the `wlconfig` task, or nested within a `create` or `query` element.

The `set` element has the following attributes:

Table 13-6 Attributes of the set Element

Attribute	Description	Data Type	Required?
attribute	The name of the MBean attribute to set.	String	Yes
value	The value to set for the specified MBean attribute. You can specify multiple object names (stored in Ant properties) as a value by delimiting the entire value list with quotes and separating the object names with a semicolon. See "Example of Setting Multiple Attribute Values" on page C-11.	String	Yes
mbean	The object name of the MBean whose values are being set. This attribute is required only when the `set` element is included as a direct child of the main `wlconfig` task; it is not required when the set element is nested within the context of a `create` or `query` element.	String	Required only when the `set` element is a direct child of the `wlconfig` task.
domain	This attribute specifies the JMX domain name for Security MBeans and third-party SPI MBeans. It is not required for administration MBeans, as the domain corresponds to the WebLogic Server domain.	String	No

get

The `get` element retrieves attribute values from an MBean in the WebLogic Server domain. The `wlconfig` task can have any number of `get` elements.

The `get` element has the following attributes.

Table 13-7 Attributes of the get Element

Attribute	Description	Data Type	Required?
attribute	The name of the MBean attribute whose value you want to retrieve.	String	Yes
property	The name of an Ant property that will hold the retrieved MBean attribute value.	String	Yes
mbean	The object name of the MBean you want to retrieve attribute values from.	String	Yes

query

The `query` elements finds MBean that match a search pattern. `query` can be used with nested `set` elements or a nested `delete` element to perform set or delete operations on all MBeans in the result set.

`wlconfig` can have any number of nested `query` elements.

`query` has the following attributes:

Table 13-8 Attributes of the query Element

Attribute	Description	Data Type	Required?
domain	The name of the WebLogic Server domain in which to search for MBeans.	String	No
type	The type of MBean to query.	String	No
name	The name of the MBean to query.	String	No
pattern	A JMX query pattern.	String	No
property	The name of an optional Ant property that will store the query results.	String	No

Table 13-8 Attributes of the query Element

Attribute	Description	Data Type	Required?
domain	This attribute specifies the JMX domain name for Security MBeans and third-party SPI MBeans. It is not required for administration MBeans, as the domain corresponds to the WebLogic Server domain.	String	No

D WebLogic SNMP Agent Command-Line Reference

WebLogic Server can use Simple Network Management Protocol (SNMP) to communicate with enterprise-wide management systems. The WebLogic Server subsystem that gathers WebLogic management data, converts it to SNMP communication modules (trap notifications), and forwards the trap notifications to third-party SNMP management systems is called the WebLogic SNMP agent. The WebLogic SNMP agent runs on the Administration Server and collects information from all Managed Servers within a domain.

The WebLogic SNMP agent provides a command-line interface that enables you to:

- Retrieve the value of WebLogic Server attributes that are exposed as managed objects in the WebLogic Server MIB.
- Generate and receive WebLogic Server traps.

The following sections describe working with the WebLogic SNMP agent through its command-line interface:

- "Required Environment and Syntax for the SNMP Command-Line Interface" on page D-2
- "Commands for Retrieving the Value of WebLogic Server Attributes" on page D-4
- "Commands for Testing Traps" on page D-11

For more information about using SNMP with WebLogic Server, refer to the WebLogic SNMP Management Guide.

Required Environment and Syntax for the SNMP Command-Line Interface

Before you use the WebLogic SNMP agent's command-line interface, set up your environment and note command syntax information as described in the following sections.

Environment

To set up your environment for the WebLogic SNMP agent's command-line interface:

1. Install and configure the WebLogic Server software, as described in the *WebLogic Server Installation Guide.* See `http://e-docs.bea.com/wls/docs70/install/index.html`.
2. Enable the WebLogic SNMP agent, as described in "Setting Up the WebLogic SNMP Agent" in the *Administration Console Online Help*.

   Note: The `snmpv1trap` and `snmptrapd` commands do not require the SNMP agent to be enabled.
3. Open a command prompt (shell) and invoke the following script

   `WL_HOME\server\bin\setWLSEnv.sh` (or `setWLSEnv.cmd` on Windows)

   where `WL_HOME` is the directory in which you installed WebLogic Server.

   The script adds a supported JDK to the shell's `PATH` environment variable and adds WebLogic Server classes to the `CLASSPATH` variable.

Common Arguments

All WebLogic SNMP agent commands take the following form:

```
java command-name arguments
```

Table D-1 describes arguments that are common to most WebLogic SNMP agent commands.

Table D-1 Common Command Line Arguments

Argument	Definition
`-d`	Includes debugging information and packet dumps in the command output.
`-c` *`snmpCommunity`* `[@`*`server_name`*`|` `@`*`domain_name`*`]`	Specifies the community name that the WebLogic SNMP agent uses to secure SNMP data **and** specifies the server instance that hosts the objects with which you want to interact. To request the value of an object on the Administration Server, specify: *`snmpCommunity`* where *`snmpCommunity`* is the SNMP community name that you set in the Community Prefix field when you configured the SNMP agent (as described in "Setting Up the WebLogic SNMP Agent" in the *Administration Console Online Help*). To request the value of an object on a Managed Server, specify: *`snmpCommunity@server_name`* where *`server_name`* is the name of the Managed Server. To request the value of an object for all server instances in a domain, send a community string with the following form: *`snmpCommunity@domain_name`* If you do not specify a value, the command assumes `-c public`, which attempts to retrieve the value of an object that is on the Administration Server.
`-p` *`snmpPort`*	Specifies the port number on which the WebLogic SNMP agent listens for requests. If you do not specify a value, the command assumes `-p 161`.
`-t` *`timeout`*	Specifies the number of milliseconds the command waits to successfully connect to the SNMP agent. If you do not specify a value, the command assumes `-t 5000`.

Table D-1 Common Command Line Arguments

Argument	Definition
`-r` *`retries`*	Specifies the number of times the command retries unsuccessful attempts to connect to the SNMP agent. If you do not specify a value, the command exits on the first unsuccessful attempt.
`host`	Specifies the DNS name or IP address of the computer that hosts the WebLogic Server Administration Server, which is where the WebLogic SNMP agent runs.

Commands for Retrieving the Value of WebLogic Server Attributes

Table D-2 is an overview of commands that retrieve the value of WebLogic Server MBean attributes that are exposed in the WebLogic Server MIB.

Table D-2 Overview of Commands for Retrieving the Value of WebLogic Server Attributes

Command	Description
`snmpwalk`	Returns a recursive list of all managed objects that are below a specified node in the MIB tree. See "snmpwalk" on page D-5.
`snmpgetnext`	Returns a description of the managed object that immediately follows an OID that you specify. See "snmpgetnext" on page D-7.
`snmpget`	Returns a description of managed objects that correspond to one or more object-instance OIDs. See "snmpget" on page D-10.

snmpwalk

Returns a recursive list of all managed objects that are below a specified node in the MIB tree.

If you specify the OID for an object type, the command returns a list of all instances of that type along with all instances of any child object types.

For example, if you specify the OID for an object type that corresponds to an MBean, this command returns a description of all instances of the MBean **and** all instances of the attributes within the MBeans.

To see the WebLogic Server MIB tree, refer to the WebLogic Server SNMP MIB Reference. For more information about the structure of the MIB and its object identifiers (OIDs), refer to "SNMP MIB for WebLogic" in *WebLogic SNMP Management Guide*.

Syntax

```
java snmpwalk [-d] [-c snmpCommunity] [-p snmpPort]
             [-t timeout] [-r retries] host OID
```

Argument	Definition
OID	Specifies the object ID of the node from which you want to retrieve a recursive list of object values. Start the value with '.'; otherwise, references are assumed to be relative to the standard MIB (`.1.3.6.1.2.1`), not the WebLogic Server MIB.

For information about the command arguments that are not listed in the above table, refer to Table D-1, "Common Command Line Arguments," on page D-3.

Example

The following example returns all attributes of the `ServerRuntimeMBean` instance that is hosted on the Administration Server. Note that the OID `.1.3.6.1.4.1.140.625.360` refers to the `serverRuntimeTable` object type in the WebLogic MIB.

```
java snmpwalk localhost .1.3.6.1.4.1.140.625.360
```

If you invoke this command from a computer that is running the Examples Server, the command returns output similar to the following truncated output. Note that the output includes the full OID for each attribute instance below the `serverRuntimeTable` object.

```
Object ID:
.1.3.6.1.4.1.140.625.360.1.1.32.101.98.52.50.55.97.53.101.55.101.
56.97.51.98.97.52.99.97.57.53.100.51.51.98.102.51.98.57.48.98.51.
55
STRING: eb427a5e7e8a3ba4ca95d33bf3b90b37

Object ID:
.1.3.6.1.4.1.140.625.360.1.5.32.101.98.52.50.55.97.53.101.55.101.
56.97.51.98.97.52.99.97.57.53.100.51.51.98.102.51.98.57.48.98.51.
55
STRING: ServerRuntime:examplesServer

Object ID:
.1.3.6.1.4.1.140.625.360.1.10.32.101.98.52.50.55.97.53.101.55.101
.56.97.51.98.97.52.99.97.57.53.100.51.51.98.102.51.98.57.48.98.51
.55
STRING: ServerRuntime

Object ID:
.1.3.6.1.4.1.140.625.360.1.15.32.101.98.52.50.55.97.53.101.55.101
.56.97.51.98.97.52.99.97.57.53.100.51.51.98.102.51.98.57.48.98.51
.55
STRING: examplesServer
...
```

The following example retrieves the name of all servers in the `examples` domain. The OID specified in the example command is the numerical value that the WebLogic Server MIB assigns to the `serverRuntimeName` object type.

```
java snmpwalk -c public@examples localhost
.1.3.6.1.4.1.140.625.360.1.15
```

The following example returns all attributes of the `ServerRuntimeMBean` instance that is hosted on a Managed Server named `MS1`. Note that the OID `.1.3.6.1.4.1.140.625.360` refers to the `serverRuntimeTable` object in the WebLogic MIB.

```
java snmpwalk -c public@MS1 localhost .1.3.6.1.4.1.140.625.360
```

snmpgetnext

Returns a description of the managed object that immediately follows one or more OIDs that you specify.

Instead of the recursive listing that the `snmpwalk` command provides, this command returns the description of only the one managed object whose OID is the next in sequence. You could string together a series of `snmpgetnext` commands to achieve the same result as the `snmpwalk` command.

If you specify an object type, this command returns the first instance of the object type, regardless of how many instances of the type exist.

To see the WebLogic Server MIB tree, refer to the WebLogic Server SNMP MIB Reference. For information about the structure of the MIB and its object identifiers (OIDs), refer to "SNMP MIB for WebLogic" in *WebLogic SNMP Management Guide*.

Syntax

```
java snmpgetnext [-d] [-c snmpCommunity] [-p snmpPort]
             [-t timeout] [-r retries] host OID [OID]...
```

Argument	Definition
`OID` `[`*`OID`*`]...`	Specifies one or more object IDs. You can specify an OID for an object type or an object instance. Start the values with '.'; otherwise, references are assumed to be relative to the standard MIB (`.1.3.6.1.2.1`), not the WebLogic Server MIB.

For information about the command arguments that are not listed in the above table, refer to Table D-1, "Common Command Line Arguments," on page D-3.

Example

The following example retrieves the name of an application that has been deployed on the Administration Server. The OID in the example command is for the `applicationRuntimeName` object type, which represents the `Name` attribute of the `applicationRuntime` MBean.

```
java snmpgetnext localhost .1.3.6.1.4.1.140.625.105.1.15
```

The command returns output similar to the following:

```
Response PDU received from /127.0.0.1, community: public
Object ID:
.1.3.6.1.4.1.140.625.105.1.15.32.49.102.98.97.100.97.102.99.57.48
.50.102.48.98.53.54.100.100.49.54.50.54.99.54.99.49.97.97.98.53.
100.97
STRING: MyServer_uddiexplorer
```

To determine whether there are additional applications deployed on the Administration Server, you can use the output of the initial `snmpgetnext` command as input for an additional `snmpgetnext` command:

```
java snmpgetnext localhost
.1.3.6.1.4.1.140.625.105.1.15.32.49.102.98.97.100.97.102.99.57.48
.50.102.48.98.53.54.100.100.49.54.50.54.99.54.99.49.97.97.98.53.
100.97
```

The command returns output similar to the following:

```
Response PDU received from /127.0.0.1, community: public
Object ID:
.1.3.6.1.4.1.140.625.105.1.15.32.54.98.49.101.57.56.54.98.98.50.
57.100.54.55.48.100.56.98.101.101.97.55.48.53.57.99.49.51.56.98.
97.99
STRING: MyServer_StartupEAR
```

The following example specifies two OIDs to retrieve the name of an application that has been deployed on the Administration Server **and** the name of a JDBC connection pool. The OIDs in the example command are for the `applicationRuntimeName` object type, which represents the `Name` attribute of the `ApplicationRuntime` MBean, and `jdbcConnectionPoolRuntimeName`, which represents the `Name` attribute of the `JDBCConnectionPoolRuntimeMBean`.

```
java snmpgetnext localhost .1.3.6.1.4.1.140.625.105.1.15
 .1.3.6.1.4.1.140.625.190.1.15
```

The command returns output similar to the following:

```
Response PDU received from /127.0.0.1, community: public
Object ID:
.1.3.6.1.4.1.140.625.105.1.15.32.49.102.98.97.100.97.102.99.57.48
.50.102.48.98.53.54.100.100.49.54.50.54.99.54.99.49.97.97.98.53.
100.97
STRING: MyServer_uddiexplorer
Object ID:
.1.3.6.1.4.1.140.625.190.1.15.32.53.53.49.48.50.55.52.57.57.49.99
.102.55.48.98.53.50.54.100.48.100.53.53.52.56.49.57.49.49.99.99.
```

```
99
STRING: MyPool
```

snmpget

Retrieves the value of one or more object instances. This command does not accept OIDs for object types.

Syntax

```
java snmpget [-d] [-c snmpCommunity] [-p snmpPort]
             [-t timeout] [-r retries] host object-instance-OID
             [object-instance-OID]...
```

Argument	Definition
object-instance-OID [*object-instance-OID*]...	The object ID of an object instance. This command does not accept OIDs for object types. Start the value with '.'; otherwise, references are assumed to be relative to the standard MIB, not the WebLogic Server MIB.

Example

The following example retrieves the value of the `serverRuntimeState` and `serverRuntimeListenPort` attribute instances for the Administration Server.

```
java snmpget localhost
.1.3.6.1.4.1.140.625.360.1.60.32.102.100.48.98.101.102.100.99.102
.52.98.97.48.48.49.102.57.53.51.50.100.102.53.55.97.101.52.56.99
.99.97.99
.1.3.6.1.4.1.140.625.360.1.35.32.102.100.48.98.101.102.100.99.102
.52.98.97.48.48.49.102.57.53.51.50.100.102.53.55.97.101.52.56.99.
99.97.99
```

The command returns output similar to the following:

```
Response PDU received from /127.0.0.1, community: public
Object ID:
.1.3.6.1.4.1.140.625.360.1.60.32.102.100.48.98.101.102.100.99.102
.52.98.97.48.48.49.102.57.53.51.50.100.102.53.55.97.101.52.56.99.
99.97.99
STRING: RUNNING
Object ID:
.1.3.6.1.4.1.140.625.360.1.35.32.102.100.48.98.101.102.100.99.102
.52.98.97.48.48.49.102.57.53.51.50.100.102.53.55.97.101.52.56.99.
99.97.99
INTEGER: 7001
```

Commands for Testing Traps

Table D-3 is an overview of commands that generate and receive traps for testing purposes.

Table D-3 Overview of Commands for Retrieving Information about WebLogic Server

Command	Description
`snmpv1trap`	Constructs an SNMPv1 trap and distributes it to the SNMP manager or trap daemon that is running on the specified host and listening on the specified port number. See "snmpv1trap" on page D-12.
`snmptrapd`	Starts a daemon that receives traps and prints information about the trap. See "snmptrapd" on page D-15.

snmpv1trap

Constructs an SNMPv1 trap and distributes it to the SNMP manager or trap daemon that is running on the specified host and listening on the specified port number. For more information about the trap daemon, refer to "snmptrapd" on page D-15.

As part of invoking this command, you specify the value for fields within the trap packet that you want to send. The values that you specify must resolve to traps that are defined in the WebLogic Server MIB. For information about WebLogic Server traps and the fields that trap packets require, refer to "Format of WebLogic Trap Notifications" in the *WebLogic SNMP Management Guide*.

Syntax

```
java snmpv1trap [-d] [-c snmpCommunity] [-p TrapDestinationPort]
          TrapDestinationHost .1.3.6.1.4.140.625
          agent-addr generic-trap specific-trap timestamp
          [OID {INTEGER | STRING | GAUGE | TIMETICKS | OPAQUE |
          IPADDRESS | COUNTER} value] ...
```

Argument	Definition
`-c` *`snmpCommunity`*	Specifies a password (community name) that secures the data in the trap. If you do not specify a value, the command assumes `-c public`.
`-p` *`TrapDestinationPort`*	Specifies the port number on which the SNMP manager or trap daemon is listening. If you do not specify a value, the command assumes `-p 162`.
`TrapDestinationHost`	Specifies the DNS name or IP address of the computer that hosts the SNMP manager or trap daemon.
`.1.3.6.1.4.140.625`	Specifies the value of the trap's `enterprise` field, which contains the beginning portion of the OID for all WebLogic Server traps.
`agent-addr`	Specifies the value of the trap's `agent address` field. This field is intended to indicate the computer on which the trap was generated. When using the `snmpv1trap` command to generate a trap, you can specify any valid DNS name or IP address.

Argument	Definition
generic-trap	Specifies the value of the trap's `generic trap type` field. For a list of valid values, refer to "Format of WebLogic Trap Notifications" in the *WebLogic SNMP Management Guide*.
specific-trap	Specifies the value of the trap's `specific trap type` field. For a list of valid values, refer to "Format of WebLogic Trap Notifications" in the *WebLogic SNMP Management Guide*.
timestamp	Specifies the value of the trap's `timestamp` field. This field is intended to indicate the length of time between the last re-initialization of the SNMP agent and the time at which the trap was issued. When using the `snmpv1trap` command to generate a trap, any number of seconds is sufficient.
OID {INTEGER | STRING | GAUGE | TIMETICKS | OPAQUE | IPADDRESS | COUNTER} *value*	(Optional) Specifies the value of the trap's `variable bindings` field, which consists of name/value pairs that further describe the trap notification. For each name/value pair, specify an OID, a value type, and a value. For example, a log message trap includes a `trapTime` binding to indicate the time at which the trap is generated. To include this variable binding in the test trap that you generate, specify the OID for the `trapTime` variable binding, the `STRING` keyword, and a string that represents the time: `.1.3.6.1.4.1.140.625.100.5 STRING "2:00 pm"`

Example

The following example generates a log message trap that contains the `trapTime` and `trapServerName` variable bindings. It broadcasts the trap through port 165. In the example:

- `6` is the generic trap value that specifies "other WebLogic Server traps."
- `60` is the specific trap value that WebLogic Server uses to identify log message traps.
- `.1.3.6.1.4.1.140.625.100.5` is the OID for the `trapTime` variable binding and `.1.3.6.1.4.1.140.625.100.10` is the OID for the `trapServerName` variable binding.

```
java snmpv1trap -p 165 localhost .1.3.6.1.4.140.625 localhost 6 60
1000 .1.3.6.1.4.1.140.625.100.5 STRING "2:00 pm"
.1.3.6.1.4.1.140.625.100.10
STRING localhost
```

The SNMP manager (or trap daemon) that is listening at port number `165` receives the trap. If the trap daemon is listening on `165`, it returns the following:

```
Trap received from: /127.0.0.1, community: public
Enterprise: .1.3.6.1.4.140.625
Agent: /127.0.0.1
TRAP_TYPE: 6
SPECIFIC NUMBER: 60
Time: 1000
VARBINDS:
Object ID: .1.3.6.1.4.1.140.625.100.5
STRING: 2:00 pm
Object ID: .1.3.6.1.4.1.140.625.100.10
STRING: localhost
```

snmptrapd

Starts a daemon that receives traps and prints information about the trap.

Syntax

```
java snmpv1trap [-d] [-c snmpCommunity] [-p TrapDestinationPort]
```

Argument	Definition
`-c` *`snmpCommunity`*	Specifies that community name that the SNMP agent (or `snmpv1trap` command) used to generate the trap. If you do not specify a value, the command assumes `-c public`.
`-p` *`TrapDestinationPort`*	Specifies the port number on which the trap daemon receives traps. If you do not specify a value, the command assumes `-p 162`.

Example

The following command starts a trap daemon and instructs it to listen for requests on port 165. The daemon runs in the shell until you kill the process or exit the shell:

```
java snmptrapd -p 165
```

If the command succeeds, the trap daemon returns a blank line with a cursor. The trap daemon waits in this state until it receives a trap, at which point it prints the trap.

Example: Sending Traps to the Trap Daemon

To generate WebLogic Server traps and receive them through the trap daemon:

1. Open a command prompt (shell) and invoke the following script:

   `WL_HOME\server\bin\setWLSEnv.sh` (or `setWLSEnv.cmd` on Windows) where `WL_HOME` is the directory in which you installed WebLogic Server.

2. To start the trap daemon, enter the following command:

   ```
   java snmptrapd
   ```

3. Open another shell and invoke the following script:

   `WL_HOME\server\bin\setWLSEnv.sh` (or `setWLSEnv.cmd` on Windows)

4. To generate a trap, enter the following command:

   ```
   java snmpv1trap localhost .1.3.6.1.4.140.625 localhost 6 60
   1000
   ```

The `snmpv1trap` command generates a `serverStart` Trap and broadcasts it through port 162.

In the shell in which the trap daemon is running, the daemon prints the following:

```
Trap received from: /127.0.0.1, community: public
Enterprise: .1.3.6.1.4.140.625
Agent: /127.0.0.1
TRAP_TYPE: 6
SPECIFIC NUMBER: 60
Time: 1000
VARBINDS:
```