BEA WebLogic Server

Administration Guide

BEA WebLogic Server 6.0
Document Date: July 5, 2001

Copyright

Restricted Rights Legend

Trademarks or Service Marks

BEA WebLogic Server Administration Guide

Document Date	Software Version
July 6, 2001	BEA WebLogic Server Version 6.0

Contents

6. Deploying Applications

7. Configuring WebLogic Server Web Components

8. Deploying and Configuring Web Applications

10. Installing and Configuring the Microsoft Internet Information Server (ISAPI) Plug-In

11. Installing and Configuring the Netscape Enterprise Server Plug-In (NSAPI)

12. Managing Security

13. Managing Transactions

14. Managing JDBC Connectivity

15. Managing JMS

16. Managing JNDI

B. WebLogic Server Command-Line Interface Reference

C. Parameters for Web Server Plug-ins

Index

About This Document

This document explains the management subsystem provided for configuring and monitoring your WebLogic Server implementation. It covers the following topics:

- Chapter 1, “Overview of WebLogic Server Management,” 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, “Configuring WebLogic Servers and Clusters,” explains the facilities provided for configuring resources in a WebLogic Server domain.
- Chapter 4, “Monitoring a WebLogic Domain,” describes the facilities that are provided by WebLogic Server for monitoring the resources that make up a WebLogic Server domain.
- Chapter 5, “Using Log Messages to Manage WebLogic Servers,” describes the use of the WebLogic Server local log and the domain-wide log for managing a WebLogic Server domain.
- Chapter 6, “Deploying Applications,” describes installation of applications on the WebLogic Server and the deploying of application components.
- Chapter 7, “Configuring WebLogic Server Web Components,” explains the use of of WebLogic Server as a Web Server.
- Chapter 8, “Deploying and Configuring Web Applications,” explains deploying and configuring of Web applications.
- Chapter 9, “Installing and Configuring the Apache HTTP Server Plug-In,” explains how to install and configure the WebLogic Server Apache plug-in.

- Chapter 10, “Installing and Configuring the Microsoft Internet Information Server (ISAPI) Plug-In,” explains how to install and conifgure the WebLogic Server plug-in for the Microsoft Internet Information Server.
- Chapter 11, “Installing and Configuring the Netscape Enterprise Server Plug-In (NSAPI),” explains how to to install and configure the Netscape Enterprise Server proxy plug-in.
- Chapter 12, “Managing Security,” discusses WebLogic Server security resources and how to manage them.
- Chapter 13, “Managing Transactions,” explains how to manage the Java Transaction subsystem within a WebLogic Server domain.
- Chapter 14, “Managing JDBC Connectivity,” discusses the management of Java Database Connectivity (JDBC) resources within a WebLogic Server domain.
- Chapter 15, “Managing JMS,” discusses the management of Java Message Service within a WebLogic Server domain.
- Chapter 16, “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 17, “Managing WebLogic Server Licenses,” describes how to update your BEA license.
- Appendix A, “Using the WebLogic Server 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.

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`*`;`

Convention	Usage
UPPERCASE TEXT	Device names, environment variables, and logical operators. *Examples*: LPT1 BEA_HOME OR
`{ }`	A set of choices in a syntax line.
`[ ]`	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.

1 Overview of WebLogic Server Management

This section describes the tools available to manage WebLogic Server. This includes the following topics:

- Domains, the Administration Server and Managed Servers
- Starting the Administration Console
- Runtime and Configuration Objects
- Central Point of Access to Log Messages

Your implementation of BEA WebLogic Server™ software provides a set of interrelated resources for users. Managing these resources includes such tasks as starting and stopping servers, balancing the load on servers or connection pools, selecting and monitoring the configuration of resources, detecting and correcting problems, monitoring and evaluating system performance, and deploying Web applications, Enterprise Javabeans (EJBs) or other resources.

The main tool that WebLogic provides to accomplish these tasks is a robust, Web-based Administration Console. The Administration Console is your window into the WebLogic Administration Service. The Administration Service, implementation of Sun's Java Management Extension (JMX) standard, provides the facilities for managing WebLogic resources.

Through the Administration Console you can configure attributes of resources, deploy applications or components, monitor resource usage (such as server load or Java Virtual Machine memory usage or database connection pool load), view log messages, shutdown servers, or perform other management actions.

Domains, the Administration Server and Managed Servers

An inter-related set of WebLogic Server resources managed as a unit is called a *domain*. A domain includes one or more WebLogic Servers, and may include WebLogic Server clusters.

The configuration for a domain is defined in Extensible Markup Language (XML). Persistent storage for the domain's configuration is provided by a single XML configuration file *`install_dir`*`/config/`*`domain_name`*`/config.xml` (where *`install_dir`* is the directory under which the WebLogic Server software has been installed).

A domain is a self-contained administrative unit. If an application is deployed in a domain, components of that application cannot be deployed on servers that are not a part of that domain. When a cluster is contained in a domain, all of its servers must be a part of that domain as well.

A WebLogic Server running the Administration Service is called an *Administration Server*. The Administration Service provides the central point of control for configuring and monitoring the entire domain. The Administration Server must be running in order to perform any management operation on that domain.

In a configuration with multiple WebLogic Servers, only one server is the Administration Server; the other servers are called *Managed Servers*. Each WebLogic Managed Server obtains its configuration at startup from the Administration Server.

The same class, `weblogic.Server`, may be started as either the Administration Server for a domain or as a WebLogic Managed Server. A WebLogic Server not started as a Managed Server is an Administration Server.

In a typical configuration for a production system, the applications and components with your business logic would be deployed across Managed Servers and the role of the Administration Server would be that of configuring and monitoring the Managed Servers.

A domain is *active* if the Administration Server was started using that configuration. While the Administration Server is running, only the Administration Server can modify the configuration file. The Administration Console and the command-line administration utility provide windows into the Administration Server which enable you to modify the domain configuration.

Additional non-active configurations may reside in the configuration repository, and you can edit them using the Administration Console. The configuration repository consists of a series of subdirectories (at least one) under the `/config` directory. Each domain is defined in a distinct `config.xml` file residing in a subdirectory with the same name as the domain. To access non-active configurations, follow the `Domain Configurations` link on the Administration Console Welcome page when you start the Console.

Starting the Administration Console

The Administration Console is a Web application that uses Java Server Pages (JSPs) to access the management resources in the Administration Server.

After starting the Administration Server (see Starting and Stopping WebLogic Servers), you can start the Administration Console by directing your browser to the following URL:

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

The value of *`hostname`* is the 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`

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 you would want to ensure that requests sent to `localhost` or `127.0.0.1` or both are not sent to the proxy.

The left pane in the Administration Console contains a hierarchical tree for navigating to tables of data, configuration pages and monitoring pages, or accessing logs. By selecting (that is, left mouse clicking) an item in the domain tree, you can display a table of data for resources of a particular type (such as WebLogic Servers) or configuration and monitoring pages for a selected resource. The top-level nodes in the domain tree are containers. If leaf nodes are present in those containers, you can click on the plus sign at the left to expand the tree to access the leaf nodes.

The entity tables — tables of data about resources of a particular type — can be customized by adding or subtracting columns that display values for attributes. You can customize a table by following the `Customize this table` link at the top of the table. Each column in the table corresponds to an attribute that has been selected for inclusion in the table.

When started, the Administration Console prompts for a password. The first time the Administration Console is started, you can use the user name and password under which the Administration Server was started. You can use the Administration Console to create a list of users with administration privileges. Once designated, these users can also perform administrative tasks via the Administration Console.

Runtime and Configuration Objects

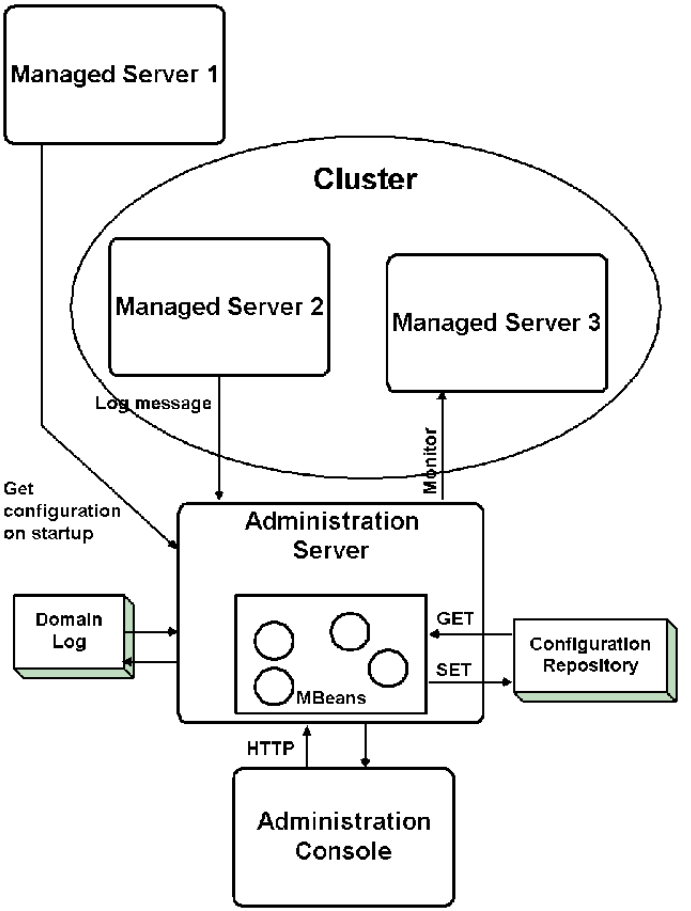
The Administration Server is populated with JavaBean-like objects called Management Beans (MBeans), which are based on Sun's Java Management Extension (JMX) standard. These objects provide management access to domain resources.

The Administration Server contains both configuration MBeans and run-time MBeans. Configuration MBeans provide both SET (write) and GET (read) access to configuration attributes.

Run-time MBeans provide a snapshot of information about domain resources, such as current HTTP sessions or the load on a JDBC connection pool. When a particular resource in the domain (such as a Web application) is instantiated, an MBean instance is created which collects information about that resource.

When you access the monitoring page for particular resources in the Administration Console, the Administration Server performs a GET operation to retrieve the current attribute values.

Figure 1-1 WebLogic Server Management Subsystem



The Administration Service allows you to change the configuration attributes of domain resources dynamically, that is, while the WebLogic Servers are running. For many attributes, you do not need to restart the servers for your change to take effect. In that case, a change in configuration is reflected in both the current run-time value of the attribute as well as the persistently stored value in the configuration file. (For more information about configuring WebLogic Servers, see Configuring WebLogic Servers and Clusters.)

In addition to the Web-based Administration Console, WebLogic Server provides a command-line utility for accessing configuration and monitoring attributes of domain resources. This tool is provided for those who want to create scripts to automate system management. (See WebLogic Server Command-Line Interface Reference.)

Central Point of Access to Log Messages

The Administration Server also provides central access to critical system messages from all the servers via the domain log. JMX provides a facility for forwarding messages to entities that subscribe for specified messages. Subscriber entities specify which messages to forward by providing a filter that selects messages of interest. A message forwarded to other network entities on the initiative of a local WebLogic Server is called a *notification*. JMX notifications are used to forward critical log messages from all WebLogic Servers in the domain to the Administration Server. When a WebLogic Managed Server starts, the Administration Server registers to receive critical log messages. Such messages are stored in the *domain log*. A single domain log filter is registered with each WebLogic Server by the Administration Server to select the messages to be forwarded. You can change the domain log filter, view the domain log, and view the local server logs using the Administration Console. (For details, see Using Log Messages to Manage WebLogic Servers.)

CHAPTER

2 Starting and Stopping WebLogic Servers

This section discusses the following topics:

- WebLogic Administration Server and WebLogic Managed Servers
- Starting the WebLogic Administration Server
- Adding a WebLogic Managed Server to the Domain
- Starting a WebLogic Managed Server
- Migrating from Earlier Versions of WebLogic Server
- Stopping WebLogic Servers from the Administration Console
- Setting up the WebLogic Server as a Windows Service
- Registering Startup and Shutdown Classes

WebLogic Administration Server and WebLogic Managed Servers

A WebLogic **domain** may consist of one or more WebLogic Servers. One (and no more than one) of these WebLogic Servers must be the Administration Server for the domain. Additional WebLogic Servers in the domain are **managed** servers. The same executable may be started as either a WebLogic Administration Server or as a WebLogic Managed Server.

Being the Administration Server is the default role for a WebLogic Server. Therefore, if there is only one WebLogic Server in a domain, that server is the Administration Server. In a multi-server domain, a WebLogic Server becomes a Managed Server only if it is instructed to obtain its configuration from a running Administration Server when started.

The Administration Server controls access to the configuration for a WebLogic domain and provides other management services such as monitoring and log message browsing. The Administration Server serves up the Administration Console which provides user access to the management services offered by the Administration Server.

When a WebLogic Managed Server is started, it obtains its configuration from the Administration Server. For this reason, booting a multi-server WebLogic domain is a two-step procedure: First you start the Administration Server, and then you start the Managed Servers.

Note: The Managed Servers must be the same WebLogic version as the Administration Server.

Startup Error Messages

When a WebLogic Server is starting, the normal logging subsystem is not yet available for logging. Accordingly, any errors encountered during startup are logged to `stdout` and to a special startup log, *`servername`*`-startup.log` (where *`servername`* is the name of the server). If startup is successful, the last message in this log points to the

location of the local server log file where normal logging occurs. For more information on the WebLogic Server logging subsystem, see Using Log Messages to Manage WebLogic Servers.

Starting the WebLogic Administration Server

There are several ways in which the WebLogic Administration Server can be started:

- From the command line

  The command to start the WebLogic Server can be either typed in a command shell manually or it can be placed in a script to avoid retyping the command each time the server is started. For information on the sample scripts provided see Starting the WebLogic Managed Servers Using Scripts.

- From the Start Menu (Windows only)
- A WebLogic Server installed as a Windows service will start automatically when the computer is rebooted.

Use of Passwords When Starting the WebLogic Server

During installation you are asked to specify a password that will be required when the server is started. If you use start scripts to start an Administration Server or a Managed Server, you can include the password as a command-line argument (See Starting the WebLogic Administration Server from the Command Line.) If you start the server using a script without the password specified as a command-line argument, you will be prompted to enter the password if there is no `password.ini` file.

Starting the WebLogic Administration Server from the Start Menu

If you installed WebLogic Server on Windows with the BEA Installation program, you can use the WebLogic Server shortcut on the Windows Start menu to start the WebLogic Administration Server. Select:

Start→Programs→BEA WebLogic E-Business Platform→Weblogic Server *Version*→Start Default Server

where *version* is the WebLogic Server software version number.

Invoking the WebLogic Server from the Start menu executes the start script `startWeblogic.cmd` (which is located in *install_dir*`/config/`*domain_name* where *domain_name* is the name of the domain and *install_dir* is the directory where you installed the WebLogic Server software).

Starting and Stopping the WebLogic Server as a Windows Service

When installed as a Windows service, the WebLogic Server starts automatically when you boot the Windows computer. The WebLogic Server is started by executing the Windows start script `startWeblogic.cmd`. A WebLogic Server started this way is started as an Administration Server. See Starting the WebLogic Administration Server from the Command Line.

To run the WebLogic Server as a Windows service, you must have installed it as such. For information on installing and removing the WebLogic Server as a Windows service, see Setting up the WebLogic Server as a Windows Service.

You can also stop and start the WebLogic Server easily from the Service Control Panel.

1. Select Start→Settings→Control Panel.
2. Double-click the Services Control Panel to open it.

3. In the Services Control Panel, scroll to the end to find `WebLogic Server`. If WebLogic is Started, you will have the option to Stop it when you select it, by clicking the Stop button to the right. If WebLogic is Stopped, the Start button will be available.

You can make the Windows service Automatic, Manual, or Disabled by clicking the Startup button and selecting a mode.

Starting the WebLogic Administration Server from the Command Line

The WebLogic Server is a Java class file, and like any Java application, you can start it with the `java` command. The arguments needed to start the WebLogic Server from the command line can be quite lengthy and typing it out whenever you need to start the server can be tedious. To make sure that your startup commands are accurate, BEA Systems recommends that you incorporate the command into a script that you can use whenever you want to start a WebLogic Server.

The following arguments are required when starting the WebLogic Administration Server from the `java` command line:

- Specify the minimum and maximum values for Java heap memory.

  For example, you may want to start the server with a default allocation of 64 megabytes of Java heap memory to the WebLogic Server. To do so, you can start the server with the `java -ms64m` and `-mx64m` options.

  These 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.

- Set the `java -classpath` option.

  The minimum content for this option is described under Setting the Classpath Option.

- Specify the name of the server.

  The domain configuration specifies configuration by server name. To specify the name of the server on the command line, use the following argument:

`-Dweblogic.Name=`*`servername`*

The default value is *`myserver`*.

- Provide user password.

  The default user is `system` and the required password is the password specified during installation. To enter the password, include the following argument:

  `-Dweblogic.management.password=`*`password`*

- Specify the WebLogic root directory if you do not start the WebLogic Server from the WebLogic root directory.

  The WebLogic root directory contains the security resources for the domain and the Configuration Repository (a directory named `\config`). You can specify the location of the root directory on the command line with the following argument:

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

  where *`path`* is the path to the root directory. If you do not specify this attribute on the command line, the current directory is used to set the runtime value of this attribute.

- Specify the location of the `bea.home` directory:

  `-Dbea.home=`*`root_install_dir`*

  where *`root_install_dir`* is the root directory under which you installed the BEA WebLogic Server software.

- If you generated a password-protected private key, you need to pass the server the private key password at startup so that the server can decrypt the PKCS private key file. To pass the private key password to the server on startup, include the following argument on the command line:

  `-Dweblogic.pkpassword=`*`pkpassword`*

  where *`pkpassword`* is the private key password.

  Password-protected private keys are generated when the Private Key Password field is specified in the Certificate Request Generator servlet. For more information, see Chapter 12, “Managing Security.”

- You can specify the name of the domain configuration when starting the Administration Server by using the following argument on the command line:

  `-Dweblogic.Domain=`*`domain_name`*

where *domain_name* is the name of the domain. This will also be the subdirectory which has the configuration file that will be used to boot the domain.

The configuration repository consists of the domains under the `/config` directory. The configuration repository may contain a variety of possible domain configurations. Each such domain is located under a separate subdirectory, with the subdirectory name being the name of that domain. When you specify *domain_name* you are thus specifying this subdirectory name. The subdirectory thus specified contains the XML configuration file (`config.xml`) and the security resources for that domain (see example below). The file `config.xml` specifies the configuration for that domain.

The domain configuration with which the Administration Server is started becomes the active domain. Only one domain can be active.

- You can also specify values for WebLogic configuration attributes on the command line. These values become the runtime value for that attribute, and any value stored in the persistent configuration is ignored. The format for setting a runtime value for a WebLogic attribute on the command line is:

  `-Dweblogic.`*`attribute=value`*

Setting the Classpath Option

The following must be included as arguments to the `-classpath` option on the `java` command line:

- *`/weblogic/`*`lib/weblogic_sp.jar`
- *`/weblogic/`*`lib/weblogic.jar`
- WebLogic Server comes with a trial version of an all-Java database management system (DBMS) called Cloudscape. If you will be using this DBMS, then you will need to include the following in the classpath:
  `/weblogic/``samples/eval/cloudscape/lib/cloudscape.jar`
- If you will be using WebLogic Enterprise Connectivity, you will need to include the following:
  `/weblogic/``lib/poolorb.jar`

where *`weblogic`* is the directory where you installed WebLogic Server.

Starting the Administration Server Using a Script

Sample scripts are provided with the WebLogic distribution that you can use to start WebLogic Servers. You will need to modify these scripts to fit your environment and applications. Separate scripts are provided for starting the Administration Server and the Managed Server. The scripts for starting the Administration Server are called `startWebLogic.sh` (UNIX) and `startWeblogic.cmd` (Windows). These scripts are located in the configuration subdirectory for your domain.

To use the supplied scripts:

- Pay close attention to classpath settings and directory names.
- Change the value of the variable `JAVA_HOME` to the location of your JDK.
- UNIX users must change the permissions of the sample UNIX script to make the file executable. For example:

  ```
  chmod +x startAdminWebLogic.sh
  ```

Restarting the Administration Server when Managed Servers are Running

For a typical production system it is recommended that the applications containing your critical business logic be deployed on Managed Servers. In such a scenario, the role of the Administration Server is that of configuring and monitoring the Managed Servers. If the Administration Server should become unavailable in such a configuration, the applications running on the Managed Servers can continue to process client requests.

When the Administration Server is started, it makes a copy of the configuration file that was used to boot the active domain. This is saved in the file

`install_dir``/config/`*`domain_name`*`/config.xml.booted`

where *`install_dir`* is the directory where you installed the WebLogic Server software and *`domain_name`* is the name of the domain. The Administration Server creates the `config.xml.booted` file only after it has successfully completed its startup sequence and is ready to process requests.

You should make a copy of this file so that you have a working configuration file that you can revert to if you need to back out of changes made to the active configuration from the Administration Console.

If the Administration Server goes down while Managed Servers continue to run, you do not need to restart the Managed Servers to recover management of the domain. The procedure for recovering management of an active domain depends upon whether you can restart the Administration Server on the same machine it was running on when the domain was started.

Restarting the Administration Server on the Same Machine

If you restart the WebLogic Administration Server while Managed Servers continue to run, the Administration Server can detect the presence of the running Managed Servers if you instruct the Administration Server to perform a discovery. To instruct the Administration Server to do a discovery of Managed Servers, enter the following argument on the command line when starting the Administration Server:

```
-Dweblogic.management.discover=true
```

The default value of this attribute is false. The configuration directory for the domain contains a file `running-managed-servers.xml` which is a list of the Managed Servers that the Administration Server knows about. When the Administration Server is instructed to perform discovery upon startup, it uses this list to check for the presence of running Managed Servers.

Restart of the Administration Server does not update the runtime configuration of the Managed Servers to take account of any changes made to attributes that can only be configured statically. WebLogic Servers must be restarted to take account of changes to static configuration attributes. Discovery of the Managed Servers does enable the Administration Server to monitor the Managed Servers or make runtime changes in attributes that can be configured dynamically.

Restarting the Administration Server on Another Machine

If a machine crash prevents you from restarting the Administration Server on the same machine, you can recover management of the running Managed Servers as follows:

1. Assign to another computer the same host name as the former Administration Server machine.

2. Install the WebLogic Server software on the new administration machine (if this has not already been done).
3. The `/config` directory (the configuration repository) used to start the Administration Server on the machine being replaced needs to be made available to the new machine. The `/config` directory could be copied from backup media or made available via NFS mount, for example. This includes the configuration file (`config.xml`) used to boot the active domain as well as applications and components installed in the `/applications` directory for that domain.
4. Restart the Administration Server on the new machine with the addition of the following argument on the command line:

   ```
   -Dweblogic.management.discover=true
   ```

   This argument will force the Administration Server to detect the presence of the Managed Servers that are running.

Adding a WebLogic Managed Server to the Domain

Before you can run a WebLogic Server as a managed server, you must first create an entry for that server in the configuration for the domain. To do this, 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. Create an entry for the server machine (Machines→Create a new machine) (if it is different than the Administration Server machine) and the new server (Servers→Create a new server).

For more information on configuring servers, see Configuring WebLogic Servers and Clusters.

Starting a WebLogic Managed Server

Once you have added WebLogic Managed Servers to your configuration (see Adding a WebLogic Managed Server to the Domain), you can start the Managed Servers from the `java` command line. The command to start the WebLogic Server can be either typed in a command shell manually or it can be placed in a script to avoid retyping the command each time the server is started. For information on the sample scripts provided see Starting the WebLogic Managed Servers Using Scripts.

The main way in which the startup parameters for a Managed Server differ from an Administration Server is that you need to provide an argument that identifies the location of the Administration Server from which the Managed Server requests its configuration. A WebLogic Server started without this parameter runs as an Administration Server.

The following are required when starting a WebLogic Managed Server:

- Specify the minimum and maximum of Java heap memory.

  For example, you may want to start the server with a default allocation of 64 megabytes of Java heap memory to the WebLogic Server. To do so, you can start the server with the `java -ms64m` and `-mx64m` options.

  These 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.

- Set the `java -classpath` option.

  The minimum content for this option is described under Setting the Classpath Option.

- Specify the name of the server.

  When a WebLogic Managed Server requests its configuration information from the Administration Server, it identifies itself to the Administration Server by server name. This enables the Administration Server to respond with the appropriate configuration for that WebLogic Server. For this reason, you must also set the server name when starting a managed server. This can be specified by adding the following argument to the command line when starting the WebLogic Managed Server:

```
-Dweblogic.Name=servername
```

- Provide the password for the `system` user.

  Only the `system` user can start a Managed Server. To specify the password for the `system` user, you can include the following argument:

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

  For information about use of passwords, see Use of Passwords When Starting the WebLogic Server.

  Note: Because `system` is the default value for the `-Dweblogic.management.username` argument, you do not need to specify it when starting a Managed Server.

- Specify the location of the `bea.home` directory:

  ```
  -Dbea.home=root_install_dir
  ```

  where *`root_install_dir`* is the root directory under which you installed the BEA WebLogic Server software.

- If you want to start the server with Secure Socket Layer (SSL) protocol, you need to pass the server the private key password at startup so that the server can decrypt the SSL private key file. To pass the SSL private key password to the server on startup, include the following argument on the command line:

  ```
  -Dweblogic.pkpassword=pkpassword
  ```

  where *`pkpassword`* is the SLL private key password.

- Specify the host name and listen port of the WebLogic Administration Server

  When starting a managed server, it is necessary to specify the host name and listen port of the Administration Server from which the managed server is to request its configuration. This can be specified by adding the following argument to the command line when starting the managed server:

  ```
  -Dweblogic.management.server=host:port
  ```

  or

  ```
  -Dweblogic.management.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 listen port. By default the Administration Server's listen port is 7001.

If you are using Secure Socket Layer (SSL) for communication with the Administration Server, the Administration Server must be specified as:

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

To use SSL protocol in communication between the Managed Servers and the Administration Server, you need to enable SSL on the Administration Server. For details on how to set this up, see Managing Security.

Note: Any WebLogic Server that is started without specifying the location of the Administration Server is started as an Administration Server.

Note: 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.

- You can also specify values for WebLogic configuration attributes on the command line. Attribute values set this way become the runtime value for that attribute, and any value stored in the persistent configuration is ignored. The format for setting a runtime value for a WebLogic attribute on the command line is:

  `-Dweblogic.`*`attribute`*`=`*`value`*

Starting the WebLogic Managed Servers Using Scripts

Sample scripts are provided with the WebLogic distribution that you can use to start WebLogic Servers. You will need to modify these scripts to fit your environment and applications. Separate scripts are provided for starting the Administration Server and the Managed Server. The scripts to start Managed Servers are called `startManagedWebLogic.sh` (UNIX) and `startManagedWebLogic.cmd` (Windows). These scripts are located in the configuration subdirectory for your domain.

To use the supplied scripts:

- Pay close attention to classpath settings and directory names.
- Change the value of the variable `JAVA_HOME` to the location of your JDK.
- UNIX users must change the permissions of the sample UNIX script to make the file executable. For example:

  ```
  chmod +x startManagedWebLogic.sh
  ```

There are two ways to start the Managed Server using the script:

- If you set the value of the environment variables `SERVER_NAME` and `ADMIN_URL`, you do not need to provide these as arguments when invoking the start script. `SERVER_NAME` should be set to the name of the WebLogic Managed Server that you wish to start. `ADMIN_URL` should be set to point to 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
  startManagedWebLogic
  ```

- You can invoke the start script and pass the name of the Managed Server and the URL for Administration Server on the command line:

  `startManagedWebLogic` *`server_name admin:url`*

  where *`server_name`* is the name of the Managed Server you are starting and *`admin_url`* is either `http://`*`host`*`:`*`port`* or `https://`*`host`*`:`*`port`* where *`host`* is the host name (or IP address) of the Administration Server and *`port`* is the port number for the Administration Server.

Migrating from Earlier Versions of WebLogic Server

If you have WebLogic Server startup scripts that you used with a previous release of the product, you will need to modify them to work with this release. The following are the main changes from previous releases:

- Dynamic class loading has changed.

  Previous releases of WebLogic Server had two separate Java classpath settings on the command line:

  - The Java system classpath
  - A special WebLogic classpath

  The WebLogic classpath property was used to facilitate dynamic class loading. In this release, the WebLogic classpath property has been eliminated and the

Java system classpath setting has changed. Scripts used with previous releases will need to be modified accordingly. In this release, dynamic loading of classes needed for Java 2 applications is the responsibility of those applications, and specifying the location of the compiled classes is accomplished via the Extensible Markup Language (XML) descriptors in the files comprising the application.

See Setting the Classpath Option for information on setting the Java system classpath.

- It is no longer necessary to specify the location of the license file or the policy file on the command line.
- The distinction between an Administration Server and Managed Servers is new to this release. You will need to add the URL to point a WebLogic Server to a running Administration Server if you want to start it as a Managed Server.
- See Starting the WebLogic Administration Server from the Command Line and Starting a WebLogic Managed Server for a complete list of the required arguments.
- New start scripts, `startManagedWebLogic.cmd` (Windows) and `startManagedWebLogic.sh` (UNIX, are provided for starting WebLogic Managed Servers. The `startWebLogic.sh` (UNIX) and `startWebLogic.cmd` (Windows) scripts are now for use in starting the WebLogic Administration Server.

Stopping WebLogic Servers from the Administration Console

To shutdown an individual WebLogic Server:

- In the Administration Console domain tree (in the left pane), select the server you want to shutdown.
- On the Monitoring→General tab page, select the **Shutdown this server** link.

Shutting Down a Server from the Command Line

You can also shut down a WebLogic Server from the command line with the following command:

```
java weblogic.Admin -url host:port SHUTDOWN -username adminname
-password password
```

where:

host is the name or IP address of the machine where the WebLogic Server is running.

port is the WebLogic Server's listen port (default is 7001).

adminname designates a user that has administrator privileges for the target WebLogic Server. Default is `system`.

password is the password for *adminname*.

Setting up the WebLogic Server as a Windows Service

You can run the WebLogic Server as a Windows service. When installed as a Windows service, the WebLogic Server starts automatically when you boot the Windows computer. A WebLogic Server is started this way by invoking the start script `startWeblogic.cmd`. Whether the WebLogic Server is started as an Administration Server or as a Managed Server depends upon the parameters in the `java` command invoking the WebLogic Server. See Starting a WebLogic Managed Server and Starting the WebLogic Administration Server from the Command Line.

To setup the WebLogic Server to run as a Windows service or to reconfigure it so it is no longer a Windows service, you must have administrator-level privileges. To install the WebLogic Server as a Windows service, do the following:

1. Navigate to the *weblogic*`\config\`*mydomain* directory (where *weblogic* is the directory where WebLogic Server was installed and *mydomain* is the subdirectory with your domain's configuration).

2. Execute the script `installNTService.cmd`.

Removing WebLogic Server as a Windows Service

To remove the WebLogic Server as a Windows service, do the following:

1. Navigate to the *`weblogic`*`\config\`*`mydomain`* directory (where *`weblogic`* is the directory where WebLogic Server was installed and *`mydomain`* is the subdirectory with your domain's configuration).
2. Execute the script `uninstallNTService.cmd`.

You can also uninstall the WebLogic Server as a Windows service from the Windows Start menu.

Changing Passwords for a Server Installed as a Windows Service

If you install the Default Server as a Windows service, the system password that you entered during installation of the WebLogic software is used when creating the service. If this password is later changed, you must do the following:

1. Uninstall the WebLogic Server as a Windows service using the `uninstallNTService.cmd` script (located in the directory *`install_dir`*`/config/`*`domain_name`* where *`install_dir`* is the directory where you installed the product).
2. The `installNTservice.cmd` script contains the following command:

   ```
   rem *** Install the service
   "C:\bea\wlserver6.0\bin\beasvc" -install -svcname:myserver
   -javahome:"C:\bea\jdk130" -execdir:"C:\bea\wlserver6.0"
   -extrapath:"C\bea\wlserver6.0\bin" -cmdline:
   %CMDLINE%
   ```

   You must append the following to the command:

   `-password:"`*`your_password`*`"`

   where *`your_password`* is the new password.

3. Execute the modified `installNTservice.cmd` script. This will create a new service with the updated password.

The WebLogic Windows Service Progam (beasvc.exe)

The scripts for installing and removing a WebLogic Server as a Windows service invoke the WebLogic Windows Service program, `beasvc.exe`. Multiple instances of WebLogic Server can be installed or removed as a Windows service using `beasvc.exe`.

All configurations for multiple services are stored in the Windows Registry using a different service name and under a server-specific hive at:

```
HKEY_LOCAL_MACHINE\SYSTEM\Current\ControlSet\Services
```

When you start the service, the Windows registry entries are picked up and the JVM is initialized and started. Since each service that is installed is independent of the others, you can install multiple instances of WebLogic Server to run as a Windows service, provided that each service is given a unique name.

The following options are available with `beasvc.exe`:

`-install`
: Install the specified service.

`-remove`
: Remove the specified service.

`-svcname:` *`service_name`*
: The user-specified name of the service to be installed or removed.

`-cmdline:` *`java_cmdline_parameters`*
: The `java` command-line parameters to be used when starting the WebLogic Server as a Windows service.

`-javahome:` *`java_directory`*
: Root directory of the Java installation. The start command will be formed by appending `\bin\java` to *`java_directory`*.

`-execdir:` *`base_dir`*
: Directory where this startup command will be executed.

`-extrapath:` *`additional_env_settings`*

Additional path settings that will be prepended to the path applicable to this command execution.

`-help`

Prints out the usage for the `beasvc.exe` command.

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. With the 1.2 or later version of the Sun Microsystems JVM, you can specify a file that contains the classpath using the @ option. You could use this option with beasvc.exe as in the following example:

```
beasvc –install –svcname:myservice -classpath:@C:\temp\myclasspath.txt
```

Registering Startup and Shutdown Classes

WebLogic provides a mechanism for performing tasks whenever a WebLogic Server starts up or gracefully shuts down. A startup class is a Java program that is automatically loaded and executed when a WebLogic Server is started or restarted. Startup classes are loaded and executed only after all other server initialization tasks have completed.

Shutdown classes work the same way as startup classes. A shutdown class is automatically loaded and executed when the WebLogic Server is shut down either from the Administration Console or using the `weblogic.admin shutdown` command.

In order for your WebLogic Servers to use startup or shutdown classes, it is necessary to register these classes, which you can do from the Administration Console.

You can register a startup or shutdown class by doing the following:

1. Access the Startup & Shutdown table from the domain tree (in the left pane) in the Administration Console. This table provides options for creating entries for shutdown or startup classes in the domain configuration.
2. Provide the class name and necessary arguments, if any, on the Configuration tab page for the startup or shutdown class you are adding.

See the Administration Console online help for more information on:

- Startup classes

- Shutdown classes

CHAPTER

3 Configuring WebLogic Servers and Clusters

This section discusses the following topics:

- Overview of Server and Cluster Configuration
- Role of the Administration Server
- Starting the Administration Console
- How Dynamic Configuration Works
- Planning A Cluster Configuration
- Server Configuration Tasks
- Cluster Configuration Tasks
- Creating a New Domain

Overview of Server and Cluster Configuration

The persistent configuration for a domain of WebLogic Servers and clusters is stored in an XML configuration file. You can modify this file in three ways:

- Through the Administration Console, BEA's graphical user interface (GUI) for managing and monitoring a domain configuration. This is intended as the main way to modify or monitor the domain configuration.
- By writing a program to modify the configuration attributes, based on the configuration Application Programmatic Interface (API) provided with WebLogic Server.
- By running the WebLogic Server command-line utility for accessing configuration attributes of domain resources. This is provided for those who want to create scripts to automate domain management.

Role of the Administration Server

IWhichever method you choose, the Administration Server must be running when you modify your domain configuration.

The Administration Server is the WebLogic Server on which the Administration Service runs. The Administration Service provides the functionality for WebLogic Server, and manages the configuration for an entire domain.

By default an instance of WebLogic Server is treated as an Administration Server. When the Administration Server starts, it loads the configuration files, which are stored, by default, in a directory called `config` under the *`WEBLOGIC_HOME`* directory. The `config` directory has a sub-directory for each domain that is available to the Administration Server. The actual configuration file resides inside the domain-specific directory and is called`config.xml`. By default, when an Administration Server starts, it looks for configuration file (`config.xml`) under the default domain directory, which is named `-mydomain`.

Each time the Administration Server is successfully started, a back up configuration file named `config.xml.booted` is created in the domain specific directory.In the unlikely event that the `config.xml` file should become corrupted during the lifetime of the server, it is possible to revert to this previously known, good configuration.

A domain may consist of only one WebLogic Server, which operates as the Administration Server.

A typical production environment contains an Administration Server and multiple WebLogic Servers. When you start the servers in such a domain, the Administration Server is started first. As each additional server is started, it is instructed to contact the Administration Server for its configuration information. In this way, the Administration Server operates as the central control entity for the configuration of the entire domain. No more than one Administration Server can be active in a domain. Only the Administration Server can modify the configuration files when it is running.

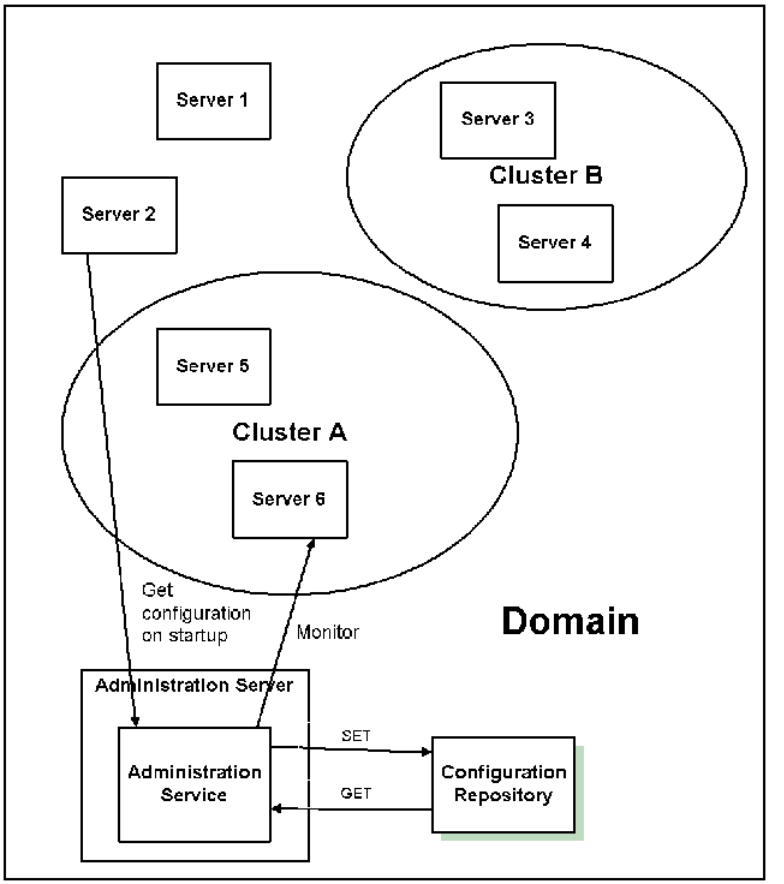


Figure 3-1 WebLogic Server Configuration

Starting the Administration Console

The main point of access to the Administration Server is through the Administration Console. To open the Administration Console, complete the following procedure:

`http://`*`host`*`:`*`port`*`/console`

In this URL, *`host`* is the host name or IP address of the machine on which the Administration Server is running and *`port`* is the address of the port at which the Administration Server is listening for requests (by default, 7001).

The system prompts you to enter a user ID and password. Enter your UserID and password. The system performs an authentication and authorization check: it verifies the user ID and password against the user database.

If you are authorized to work with the console, then the console is displayed in the access mode that the system administrator originally assigned to you: either ReadOnly or Read/Write

How Dynamic Configuration Works

WebLogic Server allows you to change the configuration attributes of domain resources dynamically, that is, while servers are running. In most cases you do not need to restart WebLogic Server for your changes to take effect. When an attribute is reconfigured, the new value is immediately reflected in both the current run-time value of the attribute and the persistent value stored in the XML configuration file.

There are exceptions, however. If, for example, you change a WebLogic Server's listen port, the new address will not be used until the next time you start the affected server. In that case, if you modify the value, you are changing the persistent value stored in the XML file and the current run-time configuration value for the attribute may differ from that persistently stored value. The Administration Console indicates

if the persistent and runtime values for a configuration attribute are not the same using an icon which changes to an alert when the server needs to be restarted for changes to

take effect.

The console does a validation check on each attribute that users change. The errors that are supported are out-of-range errors and datatype mismatch errors. In both cases, an error popup displays telling the user that an error has occurred.

Once the Administration Console has been started, if another process captures the Listen Port assigned to the Administration Server, you should remove the process that has captured the server. If you are not able to remove the process that has captured the Listen Port assigned to the Administration Server, you must edit the `Config.XML` file to change the assigned Listen Port. For information about editing the `Config.XML` file, please see the *Configuration Reference*.

Planning A Cluster Configuration

When planning a cluster configuration, keep in mind the following constraints on the networking environment and the cluster configuration.

1. The machine(s) you will be using as WebLogic hosts for the cluster must have permanently assigned, static IP addresses. You cannot use dynamically-assigned IP addresses in a clustering environment. If the servers are behind a firewall and the clients are in front of the firewall, each server must have a *public* static IP address that can be reached by the clients.
2. All WebLogic Servers in a cluster must be located on the same local area network (LAN) and must be reachable via IP multicast.
3. All servers in a cluster must be running the same version of WebLogic Server.

Configure the Servers in your cluster to support the particular mix of services that you are offering.

- For EJBs that are using JDBC connections, all the servers that deploy a particular EJB must have the same deployment and persistence configuration. This means configuring the same JDBC connection pool on each server.

- Every machine that hosts servlets must maintain the same list of servlets with identical ACLs (access control lists).
- If your client application uses JDBC connection pools directly, you must create identical connection pools (with identical ACLs) on each WebLogic Server. This means that it must be possible to create any connection pool in use on all machines in the cluster. If, for example, you configure a pool of connections to a Microsoft SQL Server database on an NT server running WebLogic, you cannot use this connection pool in a cluster that contains any non-Windows machines (that is, any machines that cannot support a Microsoft SQL Server connection).
- Other configuration details may differ for various members in the cluster. You might, for example, configure a Solaris server to process more login requests than a small NT workstation. Such differences are acceptable. Thus, in the example given here, the *performance-specific* attributes of individual cluster members may be configured with different values, so long as the service configuration for all members is identical.In practice, this often results in WebLogic Servers in the cluster being identically configured *in all areas to do with WebLogic services, class files, and external resources* (such as databases).

Server Configuration Tasks

Server configuration tasks that can be accomplished from the Administration Console include:

- Configuring an individual server using the Server node of the Administration Console. The attributes that can be changed using this node include the Server Name, the ListenPort, and the IP Address.
- Cloning an individual server using the Server node of the Administration Console. The individual server is cloned, maintaining the attribute values in the original server and the name of the new server is set on the Configuration portion of the Server node.
- Deleting a server using the Server node of the Administration Console. Click the delete icon for the server you want to delete. A dialog will appear asking you to

confirm the deletion of the server. Click Yes to confirm your decision to delete the server.

- Viewing a server log using the Server node of the Administration Console. Click the server you want to monitor. Click the Monitoring tab. Click the View Server Log link and monitor the server log in the right hand pane of the Administration Console.
- Viewing a server JNDI tree using the Server node of the Administration Console. Click the server you want to monitor. Click the Monitoring tab. Click the View JNDI Tree link and view the tree in the right hand pane of the Administration Console.
- Viewing server execute queues using the Server node of the Administration Console. Click the server you want to monitor. Click the Execute Queues link and view the table in the right hand pane of the Administration Console.
- Viewing server execute threads using the Server node of the Administration Console.Click the server you want to monitor. Click the Execute Queues link and view the table in the right hand pane of the Administration Console.
- Viewing server sockets using the Server node of the Administration Console. Click the server you want to monitor. Click the View Sockets link and view the table in the right hand pane of the Administration Console.
- Viewing server connections using the Server node of the Administration Console. Click the server you want to monitor. Click the View Connections link and view the table in the right hand pane of the Administration Console.
- Forcing garbage collection on a server using the Server node of the Administration Console. Click the server you want to monitor. Click the JVM tab. Click the Force Garbage Collection link. A dialog will appear to confirm that garbage collection has taken place.
- Monitoring server security using the Server node of the Administration Console. Click the server you want to monitor. Click the Monitoring tab. Click the Security tab. The security information will be displayed.
- Viewing server version using the Server node of the Administration Console. Click the server you want to monitor. Click the Version tab. The version data for this server will be displayed.

- Monitoring server clusters using the Server node of the Administration Console. Click the server you want to monitor. Click the Cluster tab. The cluster data for this server will be displayed.
- Deploying EJBs on a server using the Server node of the Administration Console. Click the server on which you want to deploy EJBs. Click the EJB you want to deploy and use the move control to move it to the Chosen column. Click Apply to save your selections.
- Monitoring all EJB deployments on a server using the Server node of the Administration Console. Click the server on which you want to monitor EJBs. Click the Monitor All EJB Deployments link to display the EJB Deployments table.
- Deploying web application components on a server using the Server node of the Administration Console. Click the server on which you want to deploy web applications. Click the web application you want to deploy and use the move control to move it to the Chosen column. Click Apply to save your selections.
- Monitoring all web application components on a server using the Server node of the Administration Console. Click the server on which you want to monitor web applications. Click the Monitor All Web Applications link to display the Web Application Deployments table.
- Deploy startup and shutdown classes on a server using the Server node of the Administration Console. Click the server on which you want to deploy startup classes. Click the startup class you want to deploy and use the move control to move it to the Chosen column. Click Apply to save your selections. Use the same process to deploy shutdown classes using the Shutdown Class control.
- Assigning web servers to a server using the Server node of the Administration Console. Click a server for web-application deployment. A dialog displays in the right-hand pane showing the tabs associated with this instance. Click one or more web applications in the Available column that you want to deploy on the server and use the mover control to move the web application you selected to the Chosen column. Click Apply to save your assignments.
- Assigning JDBC connection pools to a server using the Server node of the Administration Console. Click a server for web-server assignment. Click one or more JDBC connection pools in the Available column that you want to assign to the server and use the mover control to move the JDBC connection pools you selected to the Chosen column. Click Apply to save your assignments.

- Monitoring all JDBC connection pools on a server using the Server node of the Administration Console. Click a server for JDBC connection-pool monitoring. Click the Monitor All JDBC Connection Pools on This Server text link. The JDBC connection pools table displays in the right-hand pane showing all the connection pools assigned to this server.

- Assigning WLEC connection pools to a server using the Server node of the Administration Console. Click a server for WLEC connection-pool assignment. Click one or more WLEC connection pools in the Available column that you want to assign to the server and use the mover control to move the WLEC connection pools you selected to the Chosen column.

- Monitoring all WLEC connection pools on a server using the Server node of the Administration Console. Click a server for WLEC connection-pool monitoring. Click the Monitor All WLEC Connection Pools on This Server text link on the WLEC tab. The WLEC Connection Pools table displays in the right-hand pane showing all the connection pools assigned to this server.

- Assigning JMS servers, connection factories, and destinations to a server using the Server node of the Administration Console. Click a server for JMS assignments. Click one or more JMS servers in the Available column that you want to assign to the server. Click the mover control to move the JMS servers you selected to the Chosen column. Repeat using the JMS Connection Factories and JMS Destinations controls to assign connection factories and destinations t

- Assigning XML registries to a server using the Server node of the Administration Console. Click a sever for XML registry assignment. Click a registry from the XML Registry drop-down list box. Click Apply to save your selection.

- Assigning mail sessions to a server using the Server node of the Administration Console. Click a server for mail session assignment. Click one or more mail sessions in the Available column that you want to assign to the server. Use the mover control to move the mail sessions you selected to the Chosen column. Click Apply to save your selections.

- Assigning File T3s to a server using the Server node of the Administration Console. Click a server for file T3 assignment. Click one or more file T3s in the Available column that you want to assign to the server. Use the mover control to move the file T3s you selected to the Chosen column. Click Apply to save your selections.

Cluster Configuration Tasks

Cluster configuration tasks that can be accomplished from the Administration Console include:

- Configuring a cluster of servers using the Cluster node of the Administration Console. The attributes that can be changed using this node include the Cluster Name, the Cluster ListenPort, and the names of the servers in the cluster.
- Cloning a cluster of servers using the Cluster node of the Administration Console. The cluster is cloned, maintaining the attribute values and individual servers in the original cluster and the name of the new cluster is set on the Configuration portion of the Server node.
- Monitoring servers in a cluster using the Cluster node of the Administration Console. Click a cluster for server monitoring. Click the Monitor Server Participation in This Cluster text link. The server table displays in the right-hand pane showing all the servers assigned to this cluster.
- Assigning servers to a cluster using the Cluster node of the Administration Console. Click a cluster for server assignment. Click one or more servers in the Available column that you want to assign to the cluster. Use the mover control to move the servers you selected to the Chosen column. Click Apply to save your selections.
- Deleting a cluster using the Cluster node of the Administration Console. Click the Delete icon in the row of the cluster you want to delete. A dialog displays in the right-hand pane asking you to confirm your deletion request. Click Yes to confirm your decision to delete the cluster.

Creating a New Domain

This section describes how to create a new domain. The configuration information for all of the WebLogic administrative domains reside in the configuration repository, which is located under the `/config` directory. Each domain has a separate subdirectory under the `/config` directory. The name of the subdirectory for a domain must be the name of that domain.

When you first install WebLogic Server software, it is recommended that you create a zip file that has a copy of the default `/mydomain` configuration directory. You should keep a copy of this zip file as a backup that you can use for creating new domains. This subdirectory contains components that are required for a working configuration, such as a `fileRealm.properties` file and a configuration file.

To create a new domain, do the following:

1. Start the Administration Server under an existing domain such as the default `mydomain`.
2. Invoke the Administration Console by pointing your browser to:

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

   where *`hostname`* is the name of the machine where you started the Administration Server and *`port`* is the Administration Server's listen port (default is 7001).
3. Select mydomain→Create or edit other domains.

   This displays the domains table.
4. Select Default→Create a new Domain.

   Enter the name of the new domain and click Create.
5. Select the new domain from the list of domains at left to make that the current domain.
6. Now you will need to create an Administration Server entry for the new domain:

   a. Select Servers→Create a new Server.

   b. Enter the name of the new Administration Server and click Create.

7. The Administration Console will have created a new subdirectory with the name of your domain and a configuration file, `config.xml`, under that subdirectory. Now you need to create an `\applications` subdirectory in that domain directory. You can create an `\applications` subdirectory in a command shell or, on Windows, by using Explorer.

8. Next, copy the Administration Console application to the new `\applications` directory that you just created. To do this, copy the file `console.war` from the `\applications` directory under `mydomain` to the new `\applications` directory.

9. The default `mydomain` directory contains start scripts for starting the WeLogic Server. For Windows installations, these are `startWebLogic.cmd` and `startManagedWebLogic.cmd`. For UNIX installations, these are `startWebLogic.sh` and `startManagedWebLogic.sh`. Copy these start scripts to the new domain directory.

10. You will need to edit the start scripts in a text editor. By default, the name of the domain is set as:

    ```
    -Dweblogic.Domain=mydomain
    ```

    Replace `mydomain` with the name of the new domain.

    By default the name of the Administration Server is set as:

    ```
    -Dweblogic.Name=MyServer
    ```

    Replace `MyServer` with the name of the new Administration Server.

11. At the end of the start script there is a `cd` command:

    ```
    cd config\mydomain
    ```

    Replace `mydomain` with the subdirectory name of the new domain. There is also a line in the start script that reads:

    ```
    echo startWebLogic.cmd must be run from the config\mydomain
    directory.
    ```

    Replace `mydomain` here with the name of the new domain.

12. Copy `SerializedSystemIni.dat` and `fileRealm.properties` from the default `mydomain` directory to your new domain directory. Do not boot the new Administration Server before copying these files.

13. If you created a `password.ini` file during installation, you must also copy the `password.ini` file from the default `mydomain` directory to the directory for your new domain.

Once you have completed this procedure, you can start the Administration Server for your new domain.

CHAPTER

4 Monitoring a WebLogic Domain

This section explains how to monitor your WebLogic domain, including:

- Overview of Monitoring
- Monitoring Servers
- Monitoring JDBC Connection Pools
- Summary of Monitoring Pages in the Administration Console

Overview of Monitoring

The tool for monitoring the health and performance of your WebLogic domain is the Administration Console. The Administration Console allows you to view status and statistics for WebLogic resources such as servers, HTTP, the JTA subsystem, JNDI, security, CORBA connection pools, EJB, JDBC, and JMS.

Monitoring information is presented in the right pane of the Administration Console. You access a page by selecting a container or subsystem, or a particular entity under a container, on the hierarchical domain tree, in the left pane.

The Administration Console provides three types of page that contain monitoring information:

- Monitoring tab pages for a particular entity (such as an instance of a JDBC Connection Pool or a particular server's performance)

- Tables of data about all entities of a particular type (such as the WebLogic Servers table)
- Views of the domain log and of the local server logs. For information about log messages, see Using Log Messages to Manage WebLogic Servers.

The Administration Console obtains information about domain resources from the Administration Server. The Administration Server, in turn, is populated with Management Beans (MBeans), based on Sun's Java Management Extension (JMX) standard, which provides the scheme for management access to domain resources.

The Administration Server contains both configuration MBeans, which control the domain's configuration, and run-time MBeans. Run-time MBeans provide a snapshot of information about domain resources, such as JVM memory usage or the status of WebLogic Servers. When a particular resource in the domain (such as a Web application) is instantiated, an MBean instance is created which collects information about that particular resource.

When you access a monitoring page for particular resources in the Administration Console, the Administration Server performs a GET operation to retrieve the current attribute values.

The following sections describe some of the monitoring pages that are useful for managing a WebLogic domain. These pages have been selected simply to illustrate the facilities provided by the Administration Console.

Monitoring Servers

The servers table and the monitoring tab pages for individual servers enable you to monitor WebLogic Servers. The servers table provides a summary of the status of all servers in your domain. If only a small subset of the log messages from the server are forwarded to the domain log, accessing the local server log may be useful for troubleshooting or researching events.

For more information about the log files and the logging subsystem, see Using Log Messages to Manage WebLogic Servers.

You can access monitoring data for each WebLogic server from the monitoring tabs for that server. The Logging tab provides access to the local log for the server (that is, the log on the machine where the server is running).

The Monitoring→General tab page indicates the current status and provides access to the JNDI tree, the Execute Queues table, the Active Sockets table, and the Connections table. The Execute Queues table provides performance information such as the oldest pending request and the number of requests currently pending.

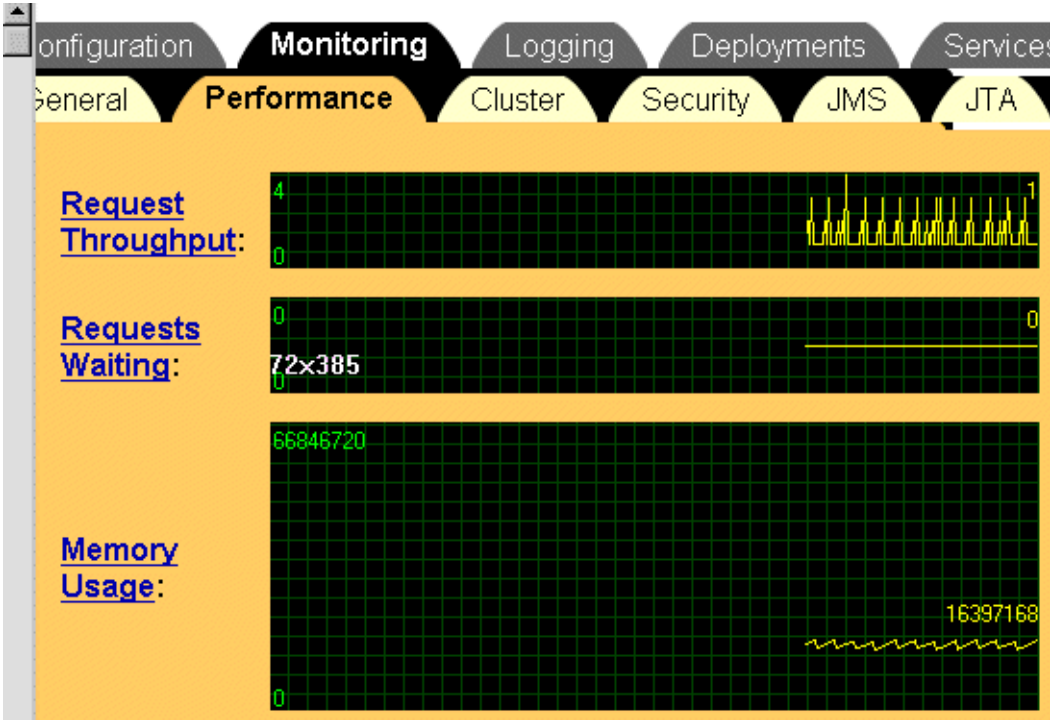
Shutting down or Suspending a Server

The Monitoring→General tab also enables you to shut down or suspend a server. If a server is suspended, it accepts requests only from the Administration Server. Client requests are ignored.

Performance

The Monitoring→Performance tab graphs real-time data on JVM memory heap usage, request throughput, and waiting requests. This tab page also enables you to force the JVM to perform garbage collection on the memory heap.

Figure 4-1 Service Performance Graphs



The Java heap is a repository for Java objects (live and dead). Normally you do not need to perform garbage collection manually because the JVM does this automatically. When the JVM begins to run out of memory, it halts all execution and uses a garbage collection algorithm to free up space no longer used by Java applications.

On the other hand, developers debugging applications may have occasion to force garbage collection manually. Manual garbage collection may be useful, for example, if they are testing for memory leaks that rapidly consume JVM memory.

Cluster Data

The Monitoring→Cluster tab provides information about the cluster that the selected server is a participant in (such as the number of servers in the cluster that are currently alive).

Server Security

The Monitoring→Security tab provides statistics about invalid login attempts and locked and unlocked users.

JMS

The Monitoring→JMS tab provides statistics on JMS servers and connections. This page also provides links to the tables of active JMS connections and active JMS servers, which monitor such attributes as total current sessions.

JTA

The Monitoring→JTA tab provides statistics on the Java Transactions subsystem such as total transactions and total rollbacks. The page provides links to tables that list transactions by resource and name, and a table of in-flight transactions.

Monitoring JDBC Connection Pools

Java Database Connectivity (JDBC) subsystem resources can also be monitored via the Administration Console. The Monitoring tab for a JDBC connection pool allows you to access a table listing statistics for the instances of that pool. As with other entity tables in the Administration Console, you can customize the table to select which attributes you want to be displayed.

A number of these attributes provide important information for managing client database access.

The Waiters High field indicates the highest number of clients waiting for a connection at one time. The Waiters field tells you how many clients are currently waiting for a connection. The Connections High field indicates the highest number of connections that have occurred at one time. The Wait Seconds High field tells you the longest duration a client has had to wait for a database connection. These attributes allow you to gauge the effectiveness of the current configuration is in responding to client requests.

If the Connections High field value is close to the value of the Maximum Capacity field (set on the Configuration Connections tab), you might consider increasing the value of Maximum Capacity (the maximum number of concurrent connections). If the value in the Waiters High field indicates that clients are subject to a long wait for database access, then you might want to increase the size of the pool.

The value in the Shrink Period field is the length of time the JDBC subsystem waits before shrinking the pool from the maximum. When the subsystem shrinks the pool, database connections are destroyed. Creating a database connection consumes resources and can be time-consuming. If your system has intermittent bursts of client requests, a short shrink period might mean that database connections are being recreated continually, which may degrade performance.

Summary of Monitoring Pages in the Administration Console

The following table lists all tables and monitoring tab pages available in the Administration Console.

Table 4-1 Summary of Monitoring Pages in the Administration Console

Page	Path to Page	Monitoring Data
Monitoring Tab Pages		
General Server Information	*servername*→Monitoring→General	State and activation time
Server Performance	*servername*→Monitoring→ Performance	Real-time graphs of request throughput, JVM memory usage, and waiting requests
Cluster Statistics	*servername*→Monitoring→Cluster	Statistics about clusters such as the number of alive servers and sent and received fragments.
Server Security	*servername*→Monitoring→Security	Number of invalid login attempts, total locked and unlocked users, and other security statistics

Table 4-1 Summary of Monitoring Pages in the Administration Console

Page	Path to Page	Monitoring Data
Server Version Information	*servername*→Monitoring→Version	JDK, WebLogic, and operating system versions
Cluster	Clusters→*clustername*→Monitoring	Information about participating servers
Entity Tables		
Servers	Servers	Server-specific data, such as memory usage, startup time, state, cluster participation, invalid login attempts, heap status, sockets counts, and total restarts
Execute Queues	*servername*→Monitoring→General→Monitor Execute Queues on this server	Information about serviced and pending requests and other attributes
Execute Sockets	*servername*→Monitoring→General→Monitor Active Sockets on this server	Protocol and other attributes of active sockets
Connections	*servername*→Monitoring→General→ Monitor Connections on this server	Connect time, remote address, bytes sent and received and other attributes of connections
Clusters	Clusters	Data such as default load algorithm and multicast address
Transactions By Name	*servername*→Monitoring→JTA→ Monitor Transactions by Name on this server	Data about transactions organized by name
Transactions By Resource	*servername*→Monitoring→JTA→ Monitor Transactions by Resource on this server	Data about transactions organized by resource
Active Transactions	*servername*→Monitoring→JTA→ Monitor In-flight Transactions on this server	Data about in-flight transactions on this server
Machines	Machines	Address and other attributes of machines
Applications	Applications	List of applications
EJB Deployments	Deployments→EJB	URL, application name, and other attributes for each EJB

Table 4-1 Summary of Monitoring Pages in the Administration Console

Page	Path to Page	Monitoring Data
Web Applications	Deployments→Web Applications	Data such as URL and default servlet for each Web application
Active Web Applications	Deployments→Web Applications→*appname*→Monitoring→ Monitor all instances of *appname*	Data about deployed copies of this Web application
Web Application Servlets	Deployments→Web Applications→*appname*→Monitoring→ Monitor all servlets for this Web Application	Statistics for the selected Web application, such as maximum pool capacity and execution time
Startup and Shutdown Classes	Deployments→Startup & Shutdown	List of registered startup and shutdown classes
JDBC Connection Pools	Services→JDBC→Connection Pools	Initial capacity, capacity increment and other attributes of JCBC Connection Pools
JDBC Multipools	Services→JDBC→Multipools	Load balancing and other attributes of JDBC Multipools
JDBC Data Sources	Services→JDBC→Data Sources	Pool name, JNDI name and other attributes of JDBC data sources
JDBC Tx Data Sources	Services→JDBC→Tx Data Sources	Pool name, JNDI name and other attributes of JDBC Tx data sources
JMS Connection Factories	Services→JMS →Connection Factories	JNDI name, client ID, default priority and other attributes of JMS connection factories
JMS Templates	Services→JMS→Templates	Data about JMS templates
JMS Destination Keys	Services→JMS→Destination Keys	Key type and other attributes of JMS destination keys
JMS Stores	Services→JMS→Stores	Descriptions of JMS stores
JMS Servers	Services→JMS→Servers	Data about JMS servers

Table 4-1 Summary of Monitoring Pages in the Administration Console

Page	Path to Page	Monitoring Data
Active JMS Services	Services→JMS→Servers→Monitor all Active JMS Services	High water mark of connections and other data about active JMS services
Active JMS Servers	Services→JMS→Servers→Monitor all instances	Statistics about sessions, messages pending, and other data
Active JMS Destinations	Services→JMS→Servers→Monitor all Active JMS Destinations	Consumers, messages received and other attributes of active JMS destinations
Active JMS Session Pools	Services→JMS→Servers→Monitoring →Monitor all Active JMS Session Pools	High water mark of consumers and other monitoring data
JMS Destinations	Services→JMS→*jmsservername*→ Destinations	JNDI name and other data
JMS Session Pools	Services→JMS→*jmsservername*→ Session Pools	Acknowledge mode, maximum sessions, and other attributes of JMS session pools
XML Registries	Services→XML→XML Registries	Lists of DocumentBuilderFactories and SAXParserFactories
WLEC Connection Pools	Services→WLEC→WLEC Connection Pools	WebLogic Enterprise (WLE) domain name, failover addresses, maximum and minimum pool size, and other information
Jolt Connection Pools	Services→Jolt	Failover addresses, maximum and minimum pool size and other attributes of Jolt connection pools
Active Jolt Connection Pools	Services→Jolt→ *joltconnectionpoolname*→ Monitoring→Monitor all active pools	Maximum capacity, current connections, and other data about instances of a Jolt connection pool
Virtual Hosts	Services→Virtual Hosts	Format, logfile name and other attributes of virtual hosts
Mail Sessions	Services→Mail	Name and properties of mail sessions
File T3	Services→File T3	Name and path of files
Users	Security→Users	List of users

Table 4-1 Summary of Monitoring Pages in the Administration Console

Page	Path to Page	Monitoring Data
Groups	Security→Groups	List of groups
Access Control Lists	Security→ACLs	List of ACLs
Caching Realms	Security→Caching Realms	Lists caching realms
Realms	Security→Realms	Describes realms
Domain Log Filters	Domain Log Filters	Servers on which the filter is registered and attributes used for filtering log messages

5 Using Log Messages to Manage WebLogic Servers

This section includes the following topics:

- Overview of Logging Subsystem
- Local Server Log Files
- Message Attributes
- Message Catalog
- Message Severity
- Browsing Log Files
- Creating Domain Log Filters

Overview of Logging Subsystem

Log messages are a useful tool for managing systems. They allow you to detect problems, track down the source of a fault, and track system performance. Log messages generated by the WebLogic Server software are stored in two locations:

- WebLogic Server component subsystems generate messages that are logged to a local file, that is, a file that resides on the machine where the server is running. If there are multiple servers on a machine, each server has its own log file. Applications deployed on your WebLogic Servers may also log messages to the server's local log file.
- In addition, a subset of messages logged locally are stored in a central domain-wide log file maintained by the Administration Server.

Java Management Extension (JMX) facilities, embedded in the WebLogic Server, are used to transmit log messages from WebLogic Servers to the Administration Server. A message forwarded to other entities on the initiative of a local WebLogic Server is called a *notification* in JMX terminology.

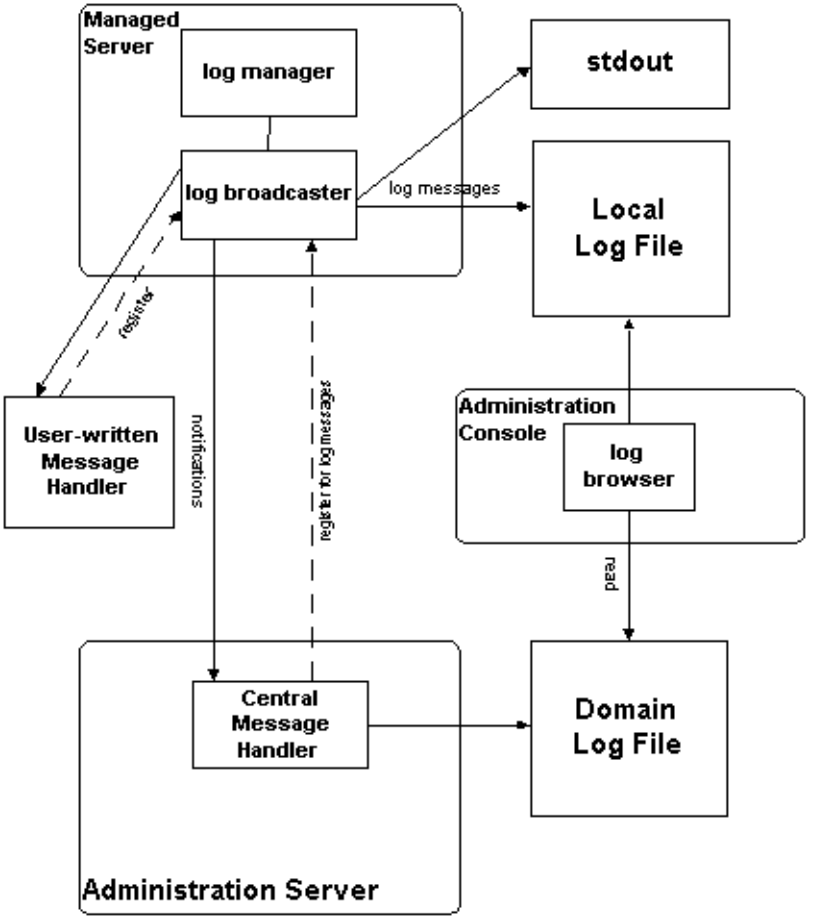
When a WebLogic server starts, the Administration Server's message handler registers with that server to receive log messages. At the time of registration, a user-modifiable filter is provided that is used by the local server to select the messages to be forwarded to the Administration Server. These messages are collected in the domain log.

By default, only the most important log messages (as determined by Message Severity) are forwarded from the local servers to the domain log. The domain log gives you an overall view of the entire domain while focusing on just the most critical messages.

If you want to modify the filter, to receive a different subset of logged messages from a local server, you can do so dynamically, using the Administration Console. You do not need to restart the local server for your changes to take effect. (See Creating Domain Log Filters.)

Developers can also build custom message handlers that can register with a WebLogic Server to receive log messages via JMX notifications.

Figure 5-1 WebLogic Server Logging Subsystem



Local Server Log Files

In previous versions of WebLogic Server, a new log file is created once the log file reaches a maximum log file size. This type of automatic log file creation is called *log rotation*. In the current release, you have the option of basing log file rotation either on size or on time. To configure rotation, open the Administration Console and do the following:

1. In the left pane, select a server.
2. In the right pane, select Configuration→Logging.
3. In the Rotation Type field, select either time or size.

   If the value in this field is `none`, no log rotation occurs. If you base log file rotation on time, a new log file is created once the specified time interval (`File Time Span`) has elapsed.

By default, the local server log file is called *`servername`*`.log` (where *`servername`* is the name of the server) and is created in the directory where you started the WebLogic Server. You can set the file name also on the Configuration→Logging page for the server.

You can specify the maximum number of rotated files that can accummulate by setting an appropriate value for the `File Count` field. Once the number of log files reaches this number, the oldest log file is deleted each time a log file rotation occurs. The rotated log files are numbered in order of creation *`filename`*nnnnn, where *`filename`* is the name configured for the log file. For example: `weblogic.log00007`.

The local server log always has all the messages that have been logged.

Configuring logging by the local server also includes the ability to specify which messages are logged to `stdout`. You can exclude messages of lower severity by specifying the lowest severity to be logged. You can also enable or disable logging of debug messages to `stdout`.

Startup Log

When a WebLogic Server is starting, if any errors occur before initialization is complete, these errors are logged to `stdout` and to a local server startup log file called `weblogic-startup.log`. If startup is successful, the last message in this log points to the location of the local server log file where normal logging occurs.

Client Logging

Java clients that use the WebLogic logging facility may also generate log messages. However, messages logged by clients are not forwarded to the domain log. You configure logging properties of a client by entering the appropriate argument on the command line:

`-Dweblogic.log.`*`attribute`*`=`*`value`*

where *`attribute`* is any `LogMBean` attribute. By default, logging to a log file is turned off for clients and messages are logged to `stdout`. You can turn on logging to a file and set the file name by using the following argument on the command line:

`-Dweblogic.log.FileName=`*`logfilename`*

where *`logfilename`* is the name of the log file.

The following command line arguments can also be used for client logging:

`-Dweblogic.StdoutEnabled=`*`boolean`*

`-Dweblogic.StdoutDebugEnabled=`*`boolean`*

```
-Dweblogic.StdoutSeverityLevel = [64 | 32 | 16 | 8 | 4 | 2 | 1 ]
```

where *`boolean`* is either `true` or `false` and the numeric values for `StdoutSeverityLevel` correspond to the following severity levels: `INFO`(64) `WARNING`(32), `ERROR`(16), `NOTICE`(8), `CRITICAL`(4), `ALERT`(2) and `EMERGENCY`(1).

Log File Format

The first line of each message in a log file begins with #### followed by the message header. The message header provides the run-time context of the message. Each attribute of the message is contained between angle brackets.

Lines following the message body are only present for messages logging an exception and display the stack trace for the exception. 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.

The following is an example of a log message:

```
####<Jun 2, 2000 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.

Note: Log messages logged by clients do not have the attributes Server Name or Thread ID.

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

Message Attributes

Each log message saved in a server log file the attributes listed in the following table may be defined. The Message Id may also associate the message with additional attributes (such as Probable Cause and Recommended Action) contained in the Message Catalog.

Attribute	Description
Timestamp	The time and date when the message originated, in a format that is specific to the locale.

Attribute	Description
Severity	Indicates the degree of impact or seriousness of the event reported by the message. See Message Severity.
Subsystem	This attribute denotes the particular subsystem of WebLogic Server that was the source of the message. For example, EJB, RMI, JMS.
Server Name Machine Name Thread ID Transaction ID	These four attributes identify the origins of the message. Transaction ID is present only for messages logged within the context of a transaction. **Note:** Server Name and Thread ID are not present in log messages generated by a Java client and logged to a client log.
User ID	The user from the security context when the message was generated.
Message ID	A unique six-digit identifier. Message IDs through 499999 are reserved for WebLogic Server system messages.
Message Text	For WebLogic Server messages, this contains the Short Description as defined in the system message catalog. (See Message Catalog.) For other messages, this is text defined by the developer of the program.

Message Catalog

In addition to the information contained in a log message, messages generated by WebLogic Server system components (or possibly by user-written code) include additional pre-defined or canned information that is stored in a message catalog. The additional attributes stored in the message catalog are described below.

Attribute	Description
Message Body	This is a short textual description of the condition being reported. This is the same as Message Text in the message.

Attribute	Description
Message Detail	A more detailed description of the condition that the message is reporting.
Probable Cause	An explanation as to why the message was logged. The probable cause of the condition the message is reporting.
Recommended Action	A recommendation for action by the administrator to resolve or avoid the condition reported in the message.

You can access these additional message attributes from log views in the Administration Console.

Message Severity

WebLogic Server log messages have an attribute called **severity** which reflects the importance or potential impact on users of the event or condition reported in the message.

Defined severities are described below. Severities are listed in order of severity with `Emergency` being the highest severity.

Severity	Forwarded to Domain Log by Default?	Meaning
Informational	No	Used for reporting normal operations.
Warning	No	A suspicious operation or configuration has occurred but it may not have an impact on normal operation.
Error	Yes	A user error has occurred. The system or application is able to handle the error with no interruption, and limited degradation, of service.

Severity	Forwarded to Domain Log by Default?	Meaning
Notice	Yes	A warning message: A suspicious operation or configuration has occurred which may not affect the normal operation of the server.
Critical	Yes	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	Yes	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	Yes	The server is in an unusable state. This severity indicates a severe system failure or panic.

Debug Messages

Messages with a severity of debug are a special case. Debug messages are not forwarded to the domain log. Debug messages may contain detailed information about an application or the server. These messages should only occur when the application is running in debug mode.

Browsing Log Files

The log browsing capabilities of the Administration Console allow you to do the following:

- View the local log file of any server.
- View the domain-wide log file.

When viewing either the domain log or the local server log, you can:

- Select log messages to be viewed based on the time of occurrence, user ID, subsystem, message severity, or the message short description.
- View messages as they are logged, or search for past log messages.
- Select the log message attributes to be displayed in the Administration Console and the order in which the attributes are displayed.

Viewing the Logs

You can access both the domain log and the local server log files from the Administration Console. How to do these tasks is discussed in the Console online help:

- Viewing the Domain Log
- Viewing the Local Server Log

Creating Domain Log Filters

The log messages forwarded by WebLogic Servers to the domain log are, by default, a subset of messages logged locally. You can configure a log filter that selects log messages for forwarding based on message severity, subsystem, or user ID. (Debug messages are a special case and are not forwarded to the domain log.) You can create or modify domain log filters from the domain log filters table. The domain log filters table is accessible from the domain monitoring tab page. See the Administration Console online help for more information on creating domain log filers.

6 Deploying Applications

This section discusses installation and deployment of applications and application components on WebLogic Server. The topics include:

- Dynamic Deployment
- Using the Administration Console to Deploy Applications

Dynamic Deployment

If auto-deployment is enabled for the target WebLogic Server domain, when an application is copied into the `/config/`*`domain_name`*`/applications` directory of the WebLogic Administration Server, if the Administration Server is running it detects the presence of the new application and deploys it automatically on the Administration Server. (The subdirectory *`domain_name`* is the name of the WebLogic Server domain in which you are deploying this application.) This technique for deploying an application is called *dynamic deployment* and is recommended only for use while you are developing applications. Dynamic deployment is not recommended for use in a production environment. If WebLogic Server is not running when you copy the application to the `/applications` directory, the application is deployed the next time the WebLogic Server is started.

By default, auto-deployment is enabled. If you have auto-deployment disabled, you can still deploy an application or application component manually via the Administration Console. This technique is called *static deployment.*

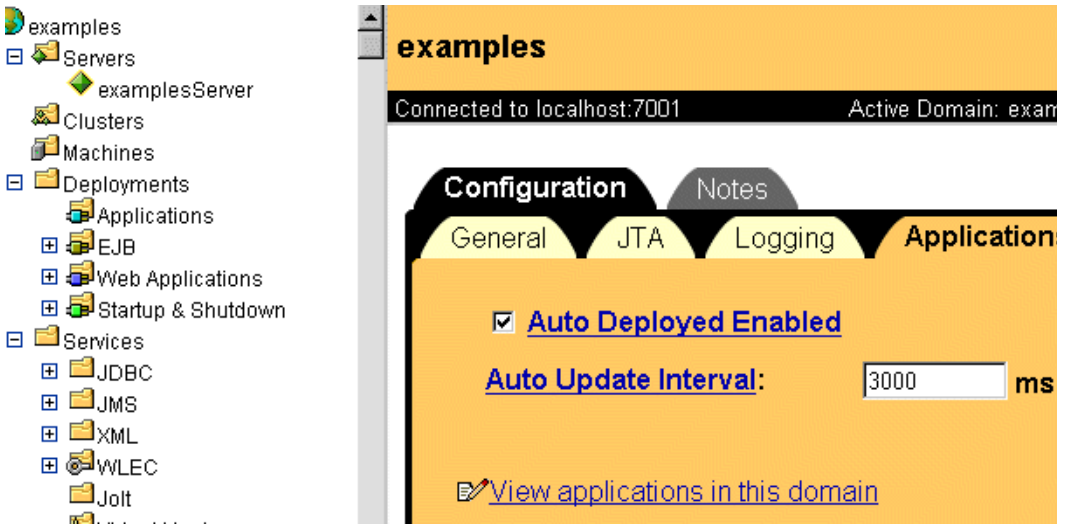
Enabling or Disabling Auto-Deployment

To determine whether you have auto-deployment enabled, invoke the Administration Console and go to the domain applications settings page (Configuration→Applications) for the domain. This page allows you to enable or disable auto-deployment and to set the interval (in milliseconds) at which the WebLogic Server checks for new applications in the `\applications` subdirectory. Figure 6-1 shows this page for the `examples` demonstration server.

By default, the Administration Server checks every three seconds for changes in the `\applications` directory when auto-deployment is enabled.

Note: Auto-deployment is a method for quickly deploying an application on the Administration Server. It is recommended that this method be used only in a development environment for testing an application. Use of auto-deployment in a production environment or for deployment of components on Managed Servers is not recommended.

Figure 6-1 Domain Applications Settings Page



Dynamic Deployment of Applications in Expanded Directory Format

An application or application component can be dynamically deployed either in expanded directory format or as packaged in an Enterprise Application Archive (EAR) file, a Web Application Archive (WAR) file, or a Java Archive (JAR) file.

To dynamically deploy an application in exploded format, do the following:

1. Make sure the directory name created for the exploded application is the same as the Context Path of the application.
2. Copy this subdirectory under `/config/`*`domain_name`*`/applications`, where *`domain_name`* is the name of the target domain where the application is to be deployed. This will automatically deploy the application if auto-deploy is enabled.

Dynamic Undeployment or Redeployment of Applications

An application or application component can be dynamically redeployed while the server is running. This may be useful if you want to update a deployed application or application component without stopping and restarting the WebLogic Administration Server. To dynamically redeploy a JAR, WAR or EAR file, simply copy the new version of the file over the existing file in the `/applications` directory.

This feature is useful for developers who can simply add the copy to the `/applications` directory as the last step in their makefile, and the server will then be updated.

Dynamic Redeployment of Exploded Applications

You can also dynamically redeploy applications or application components that have been deployed in exploded format. When an application has been deployed in exploded format, the Administration Server periodically looks for a file named `REDEPLOY` in the `WEB-INF` directory. If the timestamp on this file changes, the Administration Server redeploys the exploded directory.

If you want to update files in an exploded application directory, do the following:

1. When you first deploy the exploded application, create an empty file named `REDEPLOY` in the `WEB-INF` directory.
2. To update the exploded application, copy the updated files over the existing files in that directory.
3. After copying the new files, touch the `REDEPLOY` file in the exploded directory to alter its timestamp.

When the Administration Server detects the changed timestamp, it redeploys the contents of the exploded directory.

Using the Administration Console to Deploy Applications

You can use the Administration Console to install and deploy an application or application components (such as EJB and JSP JAR files) and deploy instances of application components on target WebLogic Servers. There are several steps to carry out this task:

1. Install the application (or application component) in the `/config/`*`domain_name`*`/applications` directory on the Administration Server (where *`domain_name`* is the name of the domain).

   Use the Install an Application page in the Administration Console to copy a J2EE application (EAR file), Web application (WAR file), or EJB or JSP (JAR file) to the `/config/`*`domain_name`*`/applications` directory on the Administration Server. There is a link to the Install an Application page from the applications table (see Figure 6-2), Web applications table and EJB deployments table. Access these tables by selecting the appropriate container in the tree in the left pane.

   Installing an application (or application component) via the Administration Console also creates entries for that application and application components in the configuration file for the domain (`/config/`*`domain_name`*`/config.xml`). The Administration Server also generates JMX Management Beans (MBeans)

that enable configuration and monitoring of the application and application components.

Figure 6-2 Applications Table for Petstore Demonstration Server

2. Deploy the application or application components.

   There are two ways to do deployment depending upon whether you have auto-deployment enabled:

   - If auto-deployment is enabled, the application is automatically deployed on the Administration Server once it is copied to the `/config/`*`domain_name`*`/applications` directory on the Administration Server.
   - If auto-deployment is disabled, an installed application is deployed only if you specify that it is to be deployed on the Configuration tab page for that application.

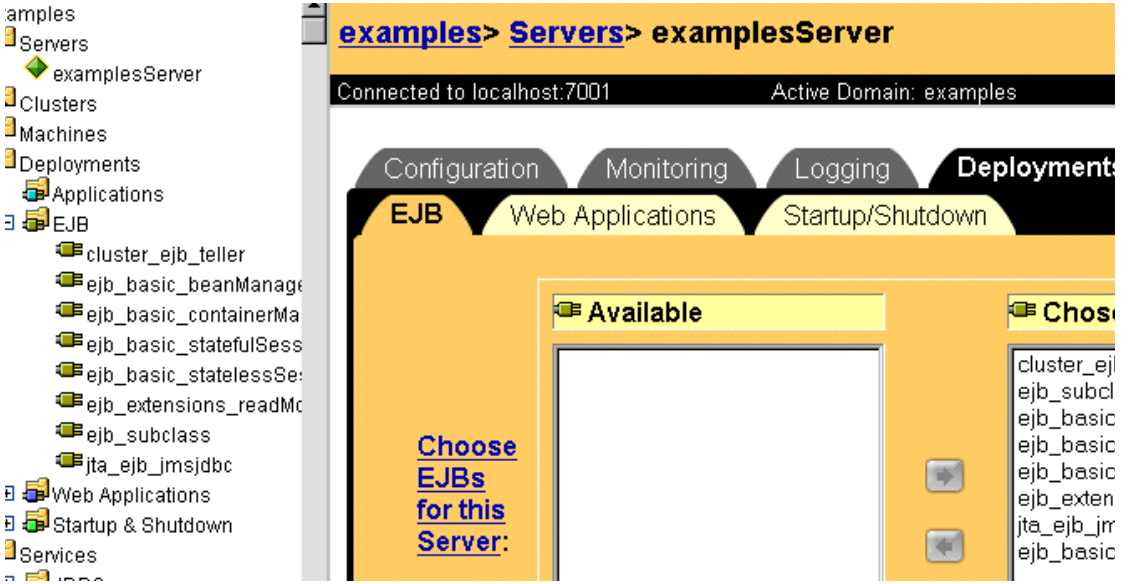
3. Deploy instances of application components (Web application components or EJBs) on Managed Servers.

   Once your application is installed on the Administration Server (in the `/config/`*`domain_name`*`/applications` directory), you can deploy components of the application to WebLogic Managed Servers.

   Select application components to be deployed on a server by accessing the Deployments→EJB (see Figure 6-3) or Deployments→Web Applications tab pages for that server.

Alternatively, you can select target servers for deployment of an application component via the Targets tab page for that component.

Figure 6-3 Deployment→EJB Tab Page for a Server



If you deploy application components (such as EJBs or WAR files) to Managed Servers in a cluster, you must ensure that the same application components are deployed on all servers in the cluster. To do this, you would select the cluster as the target for the deployment.

When an application or application component (such as an EAR or WAR file, or EJB JAR files) is deployed to a particular WebLogic Server, the files are copied to a directory `.wl_temp_do_not_delete_`*`servername`* under `/config/`*`domain_name`*`/applications` on the target WebLogic Server. The WebLogic Administration Service invokes a file distribution servlet to copy the files to the target server.

CHAPTER

7 Configuring WebLogic Server Web Components

This section discusses how to configure Web components. The following topics are covered:

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 following HTTP operating parameters using the Administration Console for each instance of WebLogic Server (or for each virtual host):

Default Web Application
: The default Web Application attempts to respond to requests that were not resolvable by any other deployed Web Application. Resources in the default Web Application are accessed with a URI that does not include the context path (the context path of a Web Application is usually the name of the Web Application).

Post Timeout Seconds
: The time (in seconds) that WebLogic Server waits between receiving chunks of data sent using the HTTP `POST` method. Used to prevent denial-of-service attacks that attempt to overload the server using the POST method.

Max Post Time
: Limits the total amount of time that WebLogic Server spends receiving `POST` data.

Max Post Size
: 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.

Enable Keep Alive
: Enables or disables persistent HTTP connections. If the browser is using HTTP 1.1, keep alive is always used.

Connection timeout
: The number of seconds that WebLogic Server waits before closing an inactive HTTP connection.

HTTPS Duration
: The number of seconds that WebLogic Server waits before closing an inactive HTTPS (Secure Socket Layer or SSL) connection.

HTTP Access Logging
: You can enable or disable the generation of HTTP access logs. You can also set parameters for when and how the access log is rotated. For more information, see "Setting Up HTTP Access Logs" on page 7-9.

For detailed information on setting these attributes, see Virtual Host at `http://e-docs.bea.com/wls/docs60/ConsoleHelp/virtualhost.html`.

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 in your Web Application. 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`.

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 Configuration/SSL tab.

Web Applications

HTTP and Web services are deployed according to the Servlet 2.2 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 form the Web Application.

For more information, see "Deploying and Configuring Web Applications" on page 8-1.

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/docs60/cluster/index.html.

Designating a Default Web Application

Every server and virtual host in your domain has a special type of Web Application, called 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 7-6)

If you do not declare a default Web Application, WebLogic Server creates a default Web Application for each server or virtual host when you start WebLogic Server. The default Web Application is named `DefaultWebApp_`*`servername`*, where *`servername`* is the name of the server you started, or in the case of a virtual host, `DefaultWebApp_`*`virtualHostName`*.

If you declare a default Web Application that fails to deploy correctly, an error is logged and the user will receive an HTTP `400` error message.

For example, if your Web Application is called `shopping`, you would use the following URI 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 URI:

```
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-hand pane, click the Web Application node
2. Select your Web Application
3. In the right-hand pane, click 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. Click the Servers (or virtual host) node in the left-hand pane.
7. Select the appropriate server or virtual host.
8. In the right-hand pane, click the General tab
9. Select the HTTP tab.
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.

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 8-17.

Setting Up a Virtual Host

To define a virtual host, use the Administration Console:

1. Create a new Virtual Host.
   a. Click on Services 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-hand 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 7-9.)

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/docs60/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-hand panel.

   b. Select the Web Application you want to target.

   c. Select the Targets tab in the right-hand 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.

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.

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. The name of the access log file includes a numeric portion that is incremented upon each rotation. Separate HTTP Access logs are kept for each Web Server you have defined.

Setting Up HTTP Access Logs by Using the Administration Console

To set up HTTP access logs use the Administration Console at `http://e-docs.bea.com/wls/docs60/ConsoleHelp/virtualhost.html`:

1. If you have set up virtual hosting:
   a. Select the services node in the left-hand pane.
   b. Select the virtual hosts node. The node expands and displays a list of virtual hosts.
   c. Select a virtual host.

   If you have *not* set up virtual hosting:

   a. Select the servers node in the left-hand pane. The node expands and displays a list of servers.

b. Select a server.

c. Select the Logging tab.

d. Select the HTTP tab.

2. Check the Enable Logging box.
3. Enter values for your Log File Name.
4. Select Common or Extended from the drop down list labeled Format.
5. Select Rotation type of `By Size` or `By Date`.
   - `By Size`: Rotates the log when it exceeds the value entered into the Log Buffer Size parameter.
   - `By Date`: Rotates the log after the number of minutes specified with the Rotation Period parameter.
6. If you have selected `Size` as the Rotation Type, in the File Min Size field specify the file size (1 - 65535 kilobytes) that triggers the server to move log messages to a separate file.

   After the log file reaches the specified size, the next time the server checks the file size, it will rename the current log file as *`FileName.n`* and create a new one to store subsequent messages.
7. Set the Flush Every parameter to the number of seconds after which the access log writes out log entries.
8. If you have selected Date as the Rotation Type, set the Rotation time to the first date when you want the log file to rotate. (Effective only if Rotation Type is set to date.) Enter the date using the `java.text.SimpleDateFormat`, MM-dd-yyyy-k:mm:ss. See the javadocs for the `java.text.SimpleDateFormat` class for more details.
9. If you have selected Date as the Rotation Type, set the Rotation Period to the number of minutes after which the log file will rotate.

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 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 step 4. in "Setting Up HTTP Access Logs by Using the Administration Console" on page 7-10).

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 7-18.

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 7-12.

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 7-16.
   - 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/docs60/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 7-12.

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 7-1 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
`String getCharacterEncoding();`	javax.servlet.ServletRequest
`int getResponseContentLength();`	`javax.servlet.ServletResponse.setContentLength()` (this method *gets* the content length of the response, as set with the *`set`*`ContentLength()` 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

Table 7-1 Getter Methods of `HttpAccountingInfo`

`HttpAccountingInfo` Methods	**Where to find information on the methods**
`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`
`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`

Table 7-1 Getter Methods of `HttpAccountingInfo`

`HttpAccountingInfo` Methods	**Where to find information on the methods**
`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`
`String getResponseHeader(String name);`	`javax.servlet.http.HttpServletResponse`

Listing 7-1 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 statefull 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 Tuning Tab located under the Configuration tab. 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.

`Tunneling Ping`
: When an HTTP tunnel connection is setup, 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 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 `400` bytes 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>
```

For more information on writing deployment descriptors see Writing Web Application Deployment Descriptors at http://e-docs.bea.com/wls/docs60/programming/webappdeployment.html.

8 Deploying and Configuring Web Applications

The following sections describes how to configure and deploy Web Applications:

Overview

A Web Application contains an application's resources such as servlets, JavaServer Pages (JSP), JSP tag libraries, and static resources such as HTML pages and image files. A Web Application can also define links to resources outside of the application such as Enterprise JavaBeans (EJB). Web Applications use a standard J2EE deployment descriptor in conjunction with a WebLogic-specific deployment descriptor to define the resources and their operating parameters.

JSP pages and HTTP servlets can access all services and APIs available in WebLogic Server. These services include EJBs, database connections via Java Database Connectivity (JDBC), JavaMessaging Service (JMS), XML, and more.

Web Applications use a standard directory structure defined in the J2EE specification and can be deployed as a collection of files that use this directory structure (this type of deployment is called *exploded directory format*) or as an archived file called a `.war` file. Deploying a Web Application in exploded directory format is recommended primarily for use while developing your application. Deploying a Web Application as a `.war` file is recommended primarily for production environments.

Steps to Deploy a Web Application

To deploy a Web Application:

1. Arrange the resources (servlets, JSPs, static files, and deployment descriptors) in the prescribed directory format. For more information, see "Directory Structure" on page 8-5.

2. Write the Web Application deployment descriptor (`web.xml`). In this step you register servlets, define servlet initialization parameters, register JSP tag libraries, define security constraints, and define other Web Application parameters. (Information on the various components of Web Applications is included throughout this document.)

   For detailed instructions, see Writing the Web Application Deployment Descriptor at `http://e-docs.bea.com/wls/docs60/programming/webappdeployment.html#web-xml`.

3. Create the WebLogic-Specific Deployment Descriptor (`weblogic.xml`). In this step you define JSP properties, JNDI mappings, security role mappings, and HTTP session parameters. If you do not need to define any of the attributes defined in this file, you do not need to create the file.

   For detailed instructions on creating the WebLogic-specific deployment descriptor, see "Writing the WebLogic-Specific Deployment Descriptor" at http://e-docs.bea.com/wls/docs60/programming/webappdeployment.html#weblogic-xml.

4. Archive the files in the above directory structure into a `.war` file. Only use archiving when the Web Application is ready for deployment in a production environment. (During development you may find it more convenient to update individual components of your Web Application by developing your application in exploded directory format.) To create a `.war` archive, use this command line from the root directory containing your Web Application:

   ```
   jar cv0f myWebApp.war .
   ```

   This command creates a Web Application archive file called `myWebApp.war`.

5. Deploy the Web Application on Weblogic Server in one of two ways: using the Administration Console or by copying the Web Application into the applications directory of your domain.

   To deploy a Web Application in archived `war` format using the Administration Console (you cannot deploy a Web Application in exploded directory format using this procedure):

   a. Select the Web Applications node in the left pane.

   b. Click Install a New Web Application.

   c. Browse to the location in your file system of the `.war` file.

   d. Click Upload.

      This procedure creates a new entry in the `config.xml` file containing the configuration for your Web Application and copies your Web Application to an internal location.

   To deploy a Web Application (in either archived or exploded format) by copying:

   a. Copy a `.war` file or the top-level directory containing a Web Application in exploded directory format into the `mydomain/config/applications` directory of your WebLogic Server distribution. (Where `mydomain` is the name of your domain.) As soon as the copy is complete, WebLogic Server automatically deploys the Web Application.

   b. (optional) Use the Administration Console to configure the Web Application. Once you change any attributes (see step 6. , below) for the Web Application, the configuration is written to the `config.xml` file and the Web Application will be statically deployed the next time you restart WebLogic Server. If you do not use the Administration Console, your Web Application is still deployed automatically every time you start WebLogic Server, even though configuration information has not been saved to the `config.xml` file.

      Note: If you deploy your Web Application in expanded form, read "Modifying Components of a Web Application" on page 8-6.

      Note: If you modify any component of a `.war` file in its original location in your file system, you must redeploy your `.war` file by uploading it again from the Administration Console.

6. Assign deployment attributes for your Web Application:

a. Open the Administration Console

b. Select the Web Applications node.

c. Select your Web Application.

d. Assign your Web Application to a WebLogic Server, cluster, or Virtual Host.

e. Select the File tab and define the appropriate attributes.

Directory Structure

You develop your Web Application within a specified directory structure so that it can be archived and deployed on WebLogic Server, or another Servlet 2.2 compliant server. All servlets, classes, static files, and other resources belonging to a Web Application are organized under a directory hierarchy. The root of this hierarchy defines the *document root* of your Web Application. All files under this root directory can be served to the client, except for files under the special directories `WEB-INF` and `META-INF` located in the root directory. Name the root directory with the name of your Web Application.This name will be used to resolve requests for components of the Web Application.

Private files should be located in the `WEB-INF` directory, under the root directory. All files under `WEB-INF` are private, and are not served to a client.

`WebApplicationName/`

Place your static files, such as HTML files and JSP files in this directory (or a subdirectory). This directory is the document root of your Web Application.

`/WEB-INF/web.xml`

The Web Application deployment descriptor that configures the Web Application.

`/WEB-INF/weblogic.xml`

The WebLogic-specific deployment descriptor file that defines how named resources in the `web.xml` file are mapped to resources residing elsewhere in WebLogic Server. This file is also used to define JSP and HTTP session attributes.

`/WEB-INF/classes`
: Contains server-side classes such as HTTP servlets and utility classes.

`/WEB-INF/lib`
: Contains `.jar` files used by the Web Application.

Deploying and Redeploying Web Applications

The procedure you use to deploy or redeploy a Web Application depend on whether the Web Application is deployed in exploded or archived format. When you modify a component of a Web Application you must also redeploy the Web Application on WebLogic Server in order to serve the modified component. These procedures are discussed in this section.

Modifying Components of a Web Application

When you modify any component of a Web Application (such as a servlet class, HTML file, JSP file, or one of the deployment descriptors), the new version of the component cannot be served by Weblogic Server until you redeploy the Web Application. The procedure used for redeployment depends on whether the Web Application is deployed as an archived `.war` file or deployed in exploded directory format.

Components in .war Format

When you edit a component of a Web Application that is deployed as a `war` file, you will need to re-jar the archive and then upload the `.war` file again.by using one of the procedures described in step 5. on page 8-4.

Components in Exploded Directory Format

When you edit a component of a Web Application that is deployed in the exploded directory format, keep in mind that WebLogic Server updates components differently:

JSP files

JSP files are redeployed based on the setting of the `pageCheckSeconds` attribute that you define in the WebLogic-specific deployment descriptor, `weblogic.xml`, of your Web Application. The attribute defines the time interval at which WebLogic Server checks JSP files to see if they have been modified. If set to `0`, pages are checked on every request. If set to `-1`, page checking and recompiling is disabled.

Note: *JSP files are redeployed automatically only to the administration server*. If you want JSPs redeployed to managed servers targeted by the Web Application, you must redeploy the Web Application. For more information, see "Redeploying a Web Application" on page 8-7.

Servlets

Servlets are redeployed based on the setting of the `Reload Period` attribute that you define in the Administration Console. Set this attribute by selecting your Web Application and then selecting the Configuration/Files tab. The attribute defines the time interval at which WebLogic Server checks servlet classes to see if they have been modified. If set to `0`, servlet classes are checked on every request. If set to `-1`, WebLogic Server does not check to see if the classes have been modified.

HTML and other static files

If you modify an HTML or other static file, such as an image file or text file, you must redeploy the Web Application in order for the changes to be recognized by WebLogic Server. Use one of the procedures below to redeploy the Web Application.

Redeploying a Web Application

Use one of the following three procedures to redeploy a Web Application:

- Use the Administration Console:

  a. Select the Web Application node.

b. Select the Web Application you want to redeploy.

c. Uncheck the Deployed box in the right-hand pane.

d. Click Apply.

e. Check the Deployed box in the right-hand pane.

f. Click Apply.

- Modify the `REDEPLOY` file:

  a. Create a sub-directory called `WEB-INF`, under the root directory of your Web Application.

  b. Create an empty text file called `REDEPLOY` and save it in the `WEB-INF` directory.

  c. To redeploy the Web Application, modify the `REDEPLOY` file by opening it, modifying the contents (adding a space character is the easiest way to do this), and then saving the file. Alternately, on UNIX machines, you can use the `touch` command on the `REDEPLOY` file.

- Re-copy a `war` into the `applications` directory (applies only to dynamic deployment). See step 5. on page 8-4.

Note: Redeploying a Web Application also redeploys the Web Application to all managed servers targeted by this Web Application.

Deploying Web Applications as Part of an Enterprise Application

You can deploy a Web Application as part of an Enterprise Application. An Enterprise Application is a J2EE deployment unit that bundles together Web Applications, EJBs, and Resource Adaptors into a single deployable unit. (For more information on Enterprise Applications, see Packaging Components and Applications at `http://e-docs.bea.com/wls/docs60/programming/packaging.html`.) If you deploy a Web Application as part of an Enterprise Application, you can specify a string that is used in place of the actual name of the Web Application when WebLogic Server resolves a request for the Web Application. You specify the new name with the `<context-root>` element in the `application.xml` deployment descriptor for the

Enterprise Application. For more information, see Client Application Deployment Descriptor Elements at `http://e-docs.bea.com/wls/docs60/programming/app_xml.html`.

For example, for a Web Application called `oranges`, you would typically request a resource from the `oranges` Web Application with a URL like:

```
http://host:port/oranges/catalog.jsp.
```

If the `oranges` Web Application is packaged in an Enterprise Application, you can specify a value for the `<context-root>` as shown in the following example:

```
<module>
   <web>
      <web-uri>oranges.war</web-uri>
      <context-root>fruit</context-root>
   </web>
</module>
```

You would then use the following URL to access the same resource from the `oranges` Web Application:

```
http://host:port/fruit/catalog.jsp
```

URIs and Web Applications

You construct the URL used to access a Web Application from a client by using the following pattern:

http://*`hoststring`*/*`ContextPath`*/*`servletPath`*/*`pathInfo`*

Where

`hoststring`
: is either a host name that is mapped to a virtual host or `hostname:portNumber`

`ContextPath`
: is the name of your Web Application

`servletPath`
: is a servlet that is mapped to the `servletPath`

pathInfo
: is the remaining portion of the URL, typically a file name.

If you are using virtual hosting, you can substitute the virtual host name for the *hoststring* portion of the URL.

For additional information, see How WebLogic Server Resolves HTTP Requests on page 8-17.

Configuring Servlets

Servlets are registered and configured as a part of a Web Application. To register a servlet, you add several entries to the Web Application deployment descriptor. The first entry, under the `<servlet>` element defines a name for the servlet and the compiled class that executes the servlet. This element also contains definitions for initialization parameters and security roles for the servlet.The second entry, under the `<servlet-mapping>` element defines the URL pattern that calls this servlet.

For complete instructions on editing the Web Application deployment descriptor, see:

- Configuring Web Applications, Deploy Servlets at `http://e-docs.bea.com/wls/docs60/programming/webappdeployment.html#servlet`
- Configuring Web Applications, Map a Servlet to a URL at `http://e-docs.bea.com/wls/docs60/programming/webappdeployment.html#servlet-mapping`

Servlet Mapping

Servlet mapping controls how you access a servlet. The following examples demonstrate some of the ways you can use servlet mapping in your Web Application. In the examples, a set of servlet configurations and mappings (from the `web.xml` deployment descriptor) is followed by a table (see "url-patterns and Servlet Invocation" on page 8-12) showing the URLs used to invoke these servlets.

Listing 8-1 Servlet Mapping Example

```
<servlet>
  <servlet-name>watermelon</servlet-name>
  <servlet-class>myservlets.watermelon</servlet-class>
</servlet>

<servlet>
  <servlet-name>garden</servlet-name>
  <servlet-class>myservlets.garden</servlet-class>
</servlet>

<servlet>
  <servlet-name>list</servlet-name>
  <servlet-class>myservlets.list</servlet-class>
</servlet>

<servlet>
  <servlet-name>kiwi</servlet-name>
  <servlet-class>myservlets.kiwi</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>watermelon</servlet-name>
  <url-pattern>/fruit/summer/*</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>garden</servlet-name>
  <url-pattern>/seeds/*</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>list</servlet-name>
  <url-pattern>/seedlist</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>kiwi</servlet-name>
  <url-pattern>*.abc</url-pattern>
</servlet-mapping>
```

Table 8-1 url-patterns and Servlet Invocation

URL	Servlet Invoked
`http://host:port/mywebapp/fruit/summer/index.html`	`watermelon`
`http://host:port/mywebapp/fruit/summer/index.abc`	`watermelon`
`http://host:port/mywebapp/seedlist`	`list`
`http://host:port/mywebapp/seedlist/index.html`	The default servlet, if configured, or an HTTP 404 file not found error message If the mapping for the `list` servlet had been `/seedlist*`, the `list` servlet would be invoked.
`http://host:port/mywebapp/seedlist/pear.abc`	`kiwi` If the mapping for the `list` servlet had been `/seedlist*`, the `list` servlet would be invoked.
`http://host:port/mywebapp/seeds`	`garden`
`http://host:port/mywebapp/seeds/index.html`	`garden`
`http://host:port/mywebapp/index.abc`	`kiwi`

Servlet Initialization Parameters

You define initialization parameters for servlets in the Web Application deployment descriptor, in the `<init-param>` element of the `<servlet>` element, using `<param-name>` and `<param-value>` tags. For example:

Listing 8-2 Example of Configuring Servlet Initialization Parameters

```
<servlet>
  <servlet-name>HelloWorld2</servlet-name>
  <servlet-class>examples.servlets.HelloWorld2</servlet-class>

  <init-param>
    <param-name>greeting</param-name>
    <param-value>Welcome</param-value>
  </init-param>

  <init-param>
    <param-name>person</param-name>
    <param-value>WebLogic Developer</param-value>
  </init-param>
</servlet>
```

For more information on editing the Web Application deployment descriptor, see Writing Web Applications Deployment Descriptors at `http://e-docs.bea.com/wls/docs60/programming/webappdeployment.html`.

Configuring JSP

You deploy JavaServer Pages (JSP) files by placing them in the root (or in a subdirectory below the root) of a Web Application. Additional JSP configuration parameters are defined in the `<jsp-descriptor>` element of the WebLogic-specific deployment descriptor, `weblogic.xml`. These parameters define the following functionality:

- Options for the JSP compiler.
- Debugging.
- How often WebLogic Server checks for updated JSPs that need to be recompiled.
- Character encoding

For a complete description of these parameters, see JSP Parameter Names and Values at `http://e-docs.bea.com/wls/docs60/programming/weblogic_xml.html#jsp-parameters`.

For instructions on editing the `weblogic.xml` file, see Creating the WebLogic-Specific Deployment Descriptor at `http://e-docs.bea.com/wls/docs60/programming/webappdeployment.html#weblogic-xml`.

Configuring JSP Tag Libraries

Weblogic Server, in accordance with the Servlet 2.2 specification provides the ability to create and use custom JSP tags. Custom JSP tags are Java classes that can be called from within a JSP page. To create custom JSP tags, you place them in a tag library and define their behavior in a tag library descriptor (TLD) file. This TLD must be made available to the Web Application containing the JSP by defining it in the Web Application deployment descriptor. It is a good idea to place the TLD file in the WEB-INF directory of your Web Application, because that directory is never available publicly.

In the Web Application deployment descriptor, you define a URI pattern for the tag library. This URI pattern must match the value in the taglib directive in your JSP pages. You also define the location of the TLD. For example, if the taglib directive in the JSP page is:

```
<%@ taglib uri="myTaglib" prefix="taglib" %>
```

and the `TLD` is located in the `WEB-INF` directory of your Web Application, you would create the following entry in the Web Application deployment descriptor:

```
<taglib>
   <taglib-uri>myTaglib</taglib-uri>
   <tablig-location>WEB-INF/myTLD.tld</taglib-location>
</taglib>
```

For more information on creating custom JSP tag libraries, see Programming JSP Tag Extensions at `http://e-docs.bea.com/wls/docs60/jsp/index.html`.

WebLogic Server also includes several custom JSP tags that you can use in your applications. These tags perform caching, facilitate query parameter-based flow control, and facilitate iterations over sets of objects. For more information, see Using Custom WebLogic JSP Tags at `http://e-docs.bea.com/wls/docs60/taglib/customtags.html`.

Configuring Welcome Pages

WebLogic Server allows you to set a page that is served by default if the requested URL is a directory. This feature can make your site easier to use, because the user can type a URL without giving a specific filename.

Welcome pages are defined at the Web Application level. If your Server is hosting multiple Web Applications, you need to define welcome pages separately for each Web Application.

To define Welcome pages, edit the Web Application deployment descriptor, `web.xml`. For more information, see Welcome Files at `http://e-docs.bea.com/wls/docs60/programming/web_xml.html#welcome-files`.

If you do not define Welcome Pages, WebLogic Server looks for the following files in the following order and serves the first one it finds:

1. `index.html`
2. `index.htm`
3. `index.jsp`

For more information, see How WebLogic Server Resolves HTTP Requests on page 8-17.

Setting Up a Default Servlet

Each Web Application has a *default servlet*. This default servlet can be a servlet that you specify, or, if you do not specify a default servlet, WebLogic Server uses an internal servlet called the `FileServlet` as the default servlet. For more information on the `FileServlet` see How WebLogic Server Resolves HTTP Requests on page 8-17.

You can register any servlet as the default servlet. Writing your own default servlet allows you to use your own logic to decide how to handle a request that falls back to the default servlet.

Setting up a default servlet replaces the `FileServlet` and should be done carefully because the `FileServlet` is used to serve most files, such as text files, HTML file, image files, and more. If you expect your default servlet to serve such files, you will need to write that functionality into your default servlet.

To set up a user-defined default servlet:

1. Define your servlet as described in Configuring Servlets on page 8-10.
2. Map your default servlet with a url-pattern of "`/`". This will cause your default servlet to respond to all types of files except for those with extensions of `*.htm` or `*.html`, which are internally mapped to the FileServlet.

   If you also want your default servlet to respond to files ending in `*.htm` or `*.html`, then you must map those extensions to your default servlet, in addition to mapping "`/`". For instructions on mapping servlets, see Configuring Servlets on page 8-10.
3. If you still want the `FileServlet` to serve files with other extensions, map those file extensions to the `FileServlet` (in addition to the mappings for your default servlet). For example, if you want the `FileServlet` to serve `gif` files, map `*.gif` to the `FileServlet`.

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 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" on page 8-8.

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 8-2 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.

Table 8-2 Examples of How WebLogic Server Resolves URLs

URL	Index Directories Checked?	This file is served in response
`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" on page 8-10.
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" on page 8-10.
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.

Table 8-2 Examples of How WebLogic Server Resolves URLs

URL	Index Directories Checked?	This file is served in response
`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.
`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" on page 8-20.
`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" on page 8-15.

Customizing HTTP Error Responses

You can configure WebLogic Server to respond with your own custom Web pages or other HTTP resources when particular HTTP errors or Java exceptions occur, instead of responding with the standard WebLogic Server error response pages.

You define custom error pages in the `<error-page>` element of the Web Application deployment descriptor (`web.xml`). For more information on error pages, see error-page Element at http://e-docs.bea.com/wls/docs60/programming/web_xml.html#error-page.

Using CGI with WebLogic Server

WebLogic Server provides functionality to support your legacy Common Gateway Interface (CGI) scripts. For new projects, we suggest you use HTTP servlets or JavaServer Pages.

WebLogic Server supports all CGI scripts via an internal WebLogic servlet called the `CGIServlet`. To use CGI, register the `CGIServlet` in the Web Application deployment descriptor (see "Example Entries to Be Included in the Web Application Deployment Descriptor when Registering the CGIServlet" on page 8-21). For more information, see Configuring Servlets on page 8-10.

Configuring WebLogic Server to use CGI

To configure CGI in WebLogic Server:

1. Declare the `CGIServlet` in your Web Application by using the `<servlet>` and `<servlet-mapping>` elements. The class name for the `CGIServlet` is `weblogic.servlet.CGIServlet`.
2. Register the following initialization parameters for the `CGIServlet` by defining the following `<init-param>` elements:

`cgiDir`

The path to the directory containing your CGI scripts. You can specify multiple directories, separated by a "`;`" (Windows) or a "`:`" (Unix). If you do not specify `cgiDir`, the directory defaults to a directory named `cgi-bin` under the Web Application root.

`extension mapping`

Maps a file extension to the interpreter or executable that runs the script. If the script does not require an executable, this initialization parameter may be omitted.

The `<param-name>` for extension mappings must begin with an asterisk followed by a dot, followed by the file extension, for example, `*.pl`.

The `<param-value>` contains the path to the interpreter or executable that runs the script You can create multiple mappings by creating a separate `<init-param>` element for each mapping.

Listing 8-3 Example Entries to Be Included in the Web Application Deployment Descriptor when Registering the CGIServlet

```
<servlet>
 <servlet-name>CGIServlet</servlet-name>
 <servlet-class>weblogic.servlet.CGIServlet</servlet-class>
 <init-param>
  <param-name>cgiDir</param-name>
  <param-value>
   /bea/wlserver6.0/config/mydomain/applications/myWebApp/cgi-bin
  </param-value>
 </init-param>

  <init-param>
   <param-name>*.pl</param-name>
   <param-value>/bin/perl.exe</param-value>
  </init-param>
</servlet>

...

<servlet-mapping>
   <servlet-name>CGIServlet</servlet-name>
   <url-pattern>/cgi-bin/*</url-pattern>
</servlet-mapping>
```

Requesting a CGI Script

The URL used to request a perl script must follow the pattern:

`http://host:port/myWebApp/cgi-bin/myscript.pl`

Where

host:port
: Is the host name and port number of WebLogic Server

cgi-bin
: is the `url-pattern` name mapped to the `CGIServlet`,

myWebApp
: is the name of your Web Application

myscript.pl
: is the name of the Perl script that is located in the directory specified by the `cgiDir` initialization parameter.

Serving Resources from the CLASSPATH with the ClasspathServlet

If you need to serve classes or other resources from the system `CLASSPATH`, or from the `WEB-INF/classes` directory of a Web Application, you can use a special servlet called the `ClasspathServlet`. The `ClasspathServlet` is useful for applications that use applets or RMI clients and require access to server-side classes. The `ClasspathServlet` is implicitly registered and available from any application.

There are two ways that you can use the `ClasspathServlet`:

- To serve a resource from the system `CLASSPATH`, call the resource with a URL such as:

  `http://server:port/classes/my/resource/myClass.class`

- To serve a resource from the `WEB-INF/classes` directory of a Web Application, call the resource with a URL such as:

  ```
  http://server:port/myWebApp/classes/my/resource/myClass.class
  ```

  In this case, the resource is located in the following directory, relative to the root of the Web Application:

  ```
  WEB-INF/classes/my/resource/myClass.class
  ```

Warning: Since the `ClasspathServlet` serves any resource located in the system `CLASSPATH`, do not place resources that should not be publicly available in the system `CLASSPATH`.

Proxying Requests to Another HTTP Server

When you use WebLogic Server as your primary Web server, you may also want to configure WebLogic Server to pass on, or proxy, certain requests to a secondary HTTP server, such as Netscape Enterprise Server, Apache, Microsoft Internet Information Server, or another instance of WebLogic Server. Any request that gets proxied is redirected to a specific URL.You can even proxy to another Web server on a different machine.You proxy requests based on the URL of the incoming request.

The `HttpProxyServlet` (provided as part of the distribution) takes an HTTP request, redirects it to the proxy URL, and sends the response to the client's browser back through WebLogic Server. To use the proxy, you must configure it in a Web Application and deploy that Web Application on the WebLogic Server that is redirecting requests.

Setting Up a Proxy to a Secondary HTTP Server

To set up a proxy to a secondary HTTP server:

1. Register the `proxy` servlet in your Web Application deployment descriptor (see “Sample web.xml for Use with ProxyServlet” on page 8-24). The Web Application must be the default Web Application of the Server that is responding to requests. The class name for the proxy servlet is `weblogic.t3.srvr.HttpProxyServlet`. For more information see Deploying

and Configuring Web Applications at
http://e-docs.bea.com/wls/docs60/adminguide/config_web_app.html
.

2. Define an initialization parameter for the `ProxyServlet` with a `<param-name>` of `redirectURL` and a `<param-value>` containing the URL of the server to which proxied requests should be directed.

3. Map the `ProxyServlet` to a `<url-pattern>`. Specifically, map the file extensions you wish to proxy, for example `*.jsp`, or `*.html`. Use the `<servlet-mapping>` element in the `web.xml` Web Application deployment descriptor.

   If you set the `<url-pattern>` to "`/`", then any request that cannot be resolved by WebLogic Server is proxied to the remote server. However, you must also specifically map the following extensions: `*.jsp`, `*.html`, and `*.html` if you want to proxy files ending with those extensions.

4. Deploy the Web Application on the WebLogic Server that redirects incoming requests.

Sample Deployment Descriptor for the Proxy Servlet

The following is an sample of a Web Applications deployment descriptor for using the Proxy Servlet.

Listing 8-4 Sample web.xml for Use with ProxyServlet

```
<! DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.
 //DTD Web Application 2.2//EN"
 "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd";>

<web-app>

<servlet>
  <servlet-name>ProxyServlet</servlet-name>
  <servlet-class>weblogic.t3.srvr.HttpProxyServlet</servlet-class
>

  <init-param>
    <param-name>redirectURL</param-name>
    <param-value>
```

```
        http://tehama:7001
      </param-value>
    </init-param>

</servlet>

<servlet-mapping>
  <servlet-name>ProxyServlet</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>ProxyServlet</servlet-name>
  <url-pattern>*.jsp</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>ProxyServlet</servlet-name>
  <url-pattern>*.htm</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>ProxyServlet</servlet-name>
  <url-pattern>*.html</url-pattern>
</servlet-mapping>

</web-app>
```

Proxying Requests to a WebLogic Cluster

The `HttpClusterServlet` (provided with the WebLogic Server distribution) proxies requests from a WebLogic Server to other WebLogic Servers in a WebLogic Cluster. The `HttpClusterServlet` provides load balancing and failover for the proxied HTTP requests. For additional information on servlets and WebLogic Clusters, see Understanding HTTP Session State Replication at `http://e-docs.bea.com/wls/docs60/cluster/servlet.html`.

Setting Up the HttpClusterServlet

To set up the `HttpClusterServlet`:

1. Configure the WebLogic Server instance that will proxy requests to a cluster of WebLogic Servers. Use the WebLogic Server Administration Console (for information on using the Administration Console see `http://e-docs.bea.com/wls/docs60/adminguide/index.html`).

   a. Create a new Web Application in your domain.

   b. Create a new Server in the domain, or use the default.

   c. Assign the Web Application you created in step a as the default Web Application for the Server that you just created.

2. Register the `HttpClusterServlet` in the Web Application deployment descriptor for the Web Application you created in step 1. (See the "Sample Deployment Descriptor for the HttpClusterServlet" on page 8-28.) The Web Application must be the default Web Application of the Server that is responding to requests. For more information see "Designating a Default Web Application" on page 7-4.

   The class name for the `HttpClusterServlet` is `weblogic.servlet.internal.HttpClusterServlet`. A Sample Deployment Descriptor for the HttpClusterServlet is included below.

3. Define the appropriate initialization parameters for the `HttpClusterServlet`. You define initialization parameters with the `<init-param> element` in the `web.xml` Web Application deployment descriptor. You must define the `defaultServers` parameter, and, where appropriate, additional parameters as described in Table 8-3 "HttpClusterServlet Parameters" on page 8-27:

4. Map the proxy servlet to a `<url-pattern>`. Specifically, map the file extensions you want to proxy, for example `*.jsp`, or `*.html`.

   If you set the url-pattern to "`/`", then any request that cannot be resolved by WebLogic Server is proxied to the remote server. However, you must also specifically map the following extensions: `*.jsp`, `*.html`, and `*.html`, if you want to proxy files ending with those extensions.

Another way to set up the url-pattern is to map a url-pattern such as `/foo` and then set the `pathTrim` parameter to `foo`, which removes `foo` from the proxied URL.

Table 8-3 HttpClusterServlet Parameters

<param-name>	<param-value>	Default Value
`defaultServers`	(Required) A list of host names and port numbers of the servers to which you are proxying requests in the form: *`host1`*`:`*`HTTP_Port`*`:`*`HTTPS_Port`*`|` *`host2`*`:`*`HTTP_Port`*`:`*`HTTPS_Port`* (Where *`host1`* and *`host2`* are the host names of servers in the cluster, *`HTTP_Port`* is the port where the host is listening for HTTP requests, and *`HTTPS_Port`* is the port where the host is litening for HTTP SSL requests.) Separate each host with the `|` character. If you set the `secureProxy` parameter to `ON` (see the `secureProxy` entry) The HTTPS port uses SSL between the WebLogic Server running `HttpClusterServlet` and the WebLogic Servers in the cluster. You must always define an HTTPS port, even if you have set `secureProxy` to `OFF`.	None
`secureProxy`	`ON/OFF`. If set to `ON`, enables SSL between the `HttpClusterServlet` and the member of a WebLogic Server cluster.	`OFF`
`DebugConfigInfo`	`ON/OFF.` If set to `on`, you can query the `HttpClusterServlet` for debugging information by adding a request parameter of `?_WebLogicBridgeConfig` to any request. For security reasons, it is recommended that you set this parameter to `OFF` in a production environment.	`OFF`

`connectionTimeout`	The amount of time, in seconds, that a socket waits in between reading chunks of data. If the timeout expires, a `java.io.InterruptedIOException` is thrown	0 = infinite timeout.
`numOfRetries`	Number of times `HttpClusterServlet` will attempt to retry a failed connection**.**	5
`pathTrim`	String to be trimmed from the beginning of the original URI.	None
`trimExt`	The file extension to be trimmed from the end of the URI.	None
`pathPrepend`	String to be prepended to the beginning of the original URL, after `pathTrim` has been trimmed, and before the request is forwarded to a WebLogic Server cluster member.	None

Sample Deployment Descriptor for the HttpClusterServlet

The following is a sample of a Web Applications deployment descriptor, `web.xml`, for using the `HttpClusterServlet`:

Listing 8-5 Sample web.xml for Use with HttpClusterServlet

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.
//DTD Web Application 2.2//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_2.dtd";>


<web-app>

<servlet>
  <servlet-name>HttpClusterServlet</servlet-name>
    <servlet-class>
      weblogic.servlet.internal.HttpClusterServlet
    </servlet-class>
```

```
  <init-param>
    <param-name>defaultServers</param-name>
    <param-value>
       myserver1:7736:7737|myserver2:7736:7737|myserver:7736:7737
    </param-value>
  </init-param>

  <init-param>
    <param-name>DebugConfigInfo</param-name>
    <param-value>ON</param-value>
  </init-param>

</servlet>

<servlet-mapping>
  <servlet-name>HttpClusterServlet</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>HttpClusterServlet</servlet-name>
  <url-pattern>*.jsp</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>HttpClusterServlet</servlet-name>
  <url-pattern>*.htm</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>HttpClusterServlet</servlet-name>
  <url-pattern>*.html</url-pattern>
</servlet-mapping>

</web-app>
```

Configuring Security in Web Applications

You can secure a Web Application by using authentication, by restricting access to certain resources in the Web Application, or by using security calls in your servlet code. Several types of security realms can be used. Security realms are discussed in the

document Security Fundamentals at http://e-docs.bea.com/wls/docs60/security/concepts.html. Note that a security realm is shared across multiple virtual hosts.

Setting Up Authentication for Web Applications

You define Authentication for Web Applications in the Web Application deployment descriptor using the `<login-config>` element. In this element you define the security realm containing the user credentials, the method of authentication, and the location of resources for authentication. For information on setting up a security realm, see Security Fundamentals at http://e-docs.bea.com/wls/docs60/security/concepts.html.

To set up authentication for Web Applications:

1. Choose an authentication method. The available options are:

   `BASIC`

   Basic authentication uses the Web Browser to display a username/password dialog box. This username and password is authenticated against the realm.

   `FORM`

   Form-based authentication requires that you return an HTML form containing the username and password. The fields returned from the form elements must be: `j_username` and `j_password`, and the action attribute must be `j_security_check`. Here is an example of the HTML coding for using FORM authentication:

   ```
   <form method="POST" action="j_security_check">
      <input type="text" name="j_username">
      <input type="password" name="j_password">
   </form>
   ```

   The resource used to generate the HTML form may be an HTML page, a JSP, or a servlet. You define this resource with the `<form-login-page>` element.

   The HTTP session object is created when the login page is served. Therefore, the `session.isNew()` method will return `FALSE` when called from pages served after successful authentication.

`CLIENT-CERT`
: Uses client certificates to authenticate the request. For more information, see Configuring the SSL Protocol at `http://e-docs.bea.com/wls/docs60/adminguide/cnfgsec.html#cnfgsec015`.

2. If you choose FORM authentication, also define the location of the resource used to generate the HTML page and a resource that responds to a failed authentication. For instructions on configuring form authentication see <login-config> at `http://e-docs.bea.com/wls/docs60/programming/web_xml.html#login-config`.

3. Define the realm used for authentication. If you do not specify a particular realm, the realm defined with the Auth Realm Name field on the Web Application→Configuration→Other tab of the Administration Console is used. For more information, see <login-config> at `http://e-docs.bea.com/wls/docs60/programming/web_xml.html#login-config`.

Multiple Web Applications, Cookies, and Authentication

By default, WebLogic Server assigns the same cookie name (`JSESSIONID`) to all Web Applications. When you use any type of authentication, all Web Applications that use the same cookie name use a single sign-on for authentication. Once a user is authenticated, that authentication will be valid for requests to any Web Application that uses the same cookie name. The user will not be prompted again for authentication.

If you want to require separate authentication for a Web Application, you can specify a unique cookie name for the Web Application. Specify the cookie name using the `CookieName` parameter, defined in the WebLogic-specific deployment descriptor `weblogic.xml`, in the `<session-descriptor>` element. For more information, see session-descriptor element at http://e-docs.bea.com/wls/docs60/programming/weblogic_xml.html#session-descriptor.

Restricting Access to Resources in a Web Application

You can apply security constraints to specified resources (servlets, JSPs, or HTML pages) in your Web Application.To apply security constraints, you

1. Define a role that is mapped to one or more principals in a security realm. You define roles with the `<security-role>` element (see http://e-docs.bea.com/wls/docs60/programming/web_xml.html#security-role) in the Web Application deployment descriptor. You then map these roles to principals in your realm with the `<security-role-assignment>` element (see http://e-docs.bea.com/wls/docs60/programming/weblogic_xml.html#security-role-assignment) in the WebLogic-specific deployment descriptor, `weblogic.xml`.
2. Define which resources in the Web Application the security constraint applies to using the `<url-pattern>` element that is nested inside the `<web-resource-collection>` element. The `<url-pattern>` can refer to either a directory, filename or a `<servlet-mapping>`.

   To apply the security constraint to the entire Web Application, use the following `<url-pattern>`:

   ```
   <url-pattern>/*</url-pattern>
   ```
3. Define the HTTP method (`GET` or `POST`) that the security constraint applies to using the `<http-method>` element that is nested inside the `<web-resource-collection>` element.
4. Define whether or not SSL should be used for communication between client and server using the `<transport-guarantee>` element nested inside of the `<user-data-constraint>` method.

Listing 8-6 Sample of Restricting Resources:

```
web.xml entries:

<security-constraint>
     <web-resource-collection>
          <web-resource-name>SecureOrdersEast</web-resource-name>
          <description>
             Security constraint for resources in the orders/east
directory
```

```
          </description>
          <url-pattern>/orders/east/*</url-pattern>
          <http-method>POST</http-method>
          <http-method>GET</http-method>
     </web-resource-collection>
     <auth-constraint>
          <description>constraint for east coast
sales</description>
          <role-name>east</role-name>
          <role-name>manager</role-name>
     </auth-constraint>
     <user-data-constraint>
          <description>SSL not required</description>
          <transport-guarantee>NONE</transport-guarantee>
     </user-data-constraint>
</security-constraint>

...

<security-role>
     <description>east coast sales</description>
     <role-name>east</role-name>
</security-role>

<security-role>
     <description>managers</description>
     <role-name>manager</role-name>
</security-role>

weblogic.xml entries:

<security-role-assignment>
     <role-name>east</role-name>
     <principal-name>tom</principal-name>
     <principal-name>jane</principal-name>
     <principal-name>javier</principal-name>
     <principal-name>maria</principal-name>
 </security-role-assignment>
 <security-role-assignment>
     <role-name> manager </role-name>
     <principal-name>peter</principal-name>
     <principal-name>georgia</principal-name>
</security-role-assignment>
```

Using Users and Roles Programmatically in Servlets

You can write your servlets to programmatically access users and roles in your servlet code using the method `javax.servlet.http.HttpServletRequest.isUserInRole(String role)`. The string `role` is mapped to the name supplied in the `<role-name>` element nested inside the `<security-role-ref>` element of a `<servlet>` declaration in the Web Application deployment descriptor. The `<role-link>` element maps to a `<role-name>` defined in the `<security-role>` element of the Web Application deployment descriptor.

For example:

Listing 8-7 Example of Security Role Mapping

```
Servlet code:
isUserInRole("manager");

web.xml entries:

<servlet>
. . .
   <role-name>manager</role-name>
   <role-link>mgr</role-link>
. . .
</servlet>

<security-role>
   <role-name>mgr</role-name>
</security-role>

weblogic.xml entries:

<security-role-assignment>
   <role-name>mgr</role-name>
   <principal-name>al</principal-name>
   <principal-name>george</principal-name>
   <principal-name>ralph</principal-name>
</security-role-ref>
```

Configuring External Resources in a Web Application

When accessing external resources, such as a DataSource from a Web Application via JNDI, you can map the JNDI name you look up in your code to the actual JNDI name as bound in the JNDI tree. This mapping is made using both the `web.xml` and `weblogic.xml` deployment descriptors and allows you to change these resources without changing your application code. You provide a name that is used in your Java code, the name of the resource as bound in the JNDI tree, and the Java type of the resource, and you indicate whether security for the resource is handled programmatically by the servlet or from the credentials associated with the HTTP request.

To configure external resources:

1. Enter the resource name in the deployment descriptor as you use it in your code, the Java type, and the security authorization type. For instructions on making deployment descriptor entries, see Reference external resources at `http://e-docs.bea.com/wls/docs60/programming/webappdeployment.html#resource-ref`.
2. Map the resource name to the JNDI name. For instructions on making deployment descriptor entries, see Map external resources at `http://e-docs.bea.com/wls/docs60/programming/webappdeployment.html#resource-description`.

This example assumes that you have defined a data source called `accountDataSource`. For more information, see JDBC Data Sources at `http://e-docs.bea.com/wls/docs60/ConsoleHelp/jdbcdatasource.html`.

Listing 8-8 Example of Using a DataSource

```
Servlet code:
javax.sql.DataSource ds = (javax.sql.DataSource) ctx.lookup
                          ("myDataSource");

web.xml entries:
```

```
<resource-ref>
. . .
   <res-ref-name>myDataSource</res-ref-name>
   <res-type>javax.sql.DataSource</res-type>
   <res-auth>CONTAINER</res-auth>
. . .
</resource-ref>

weblogic.xml entries:

<resource-description>
   <res-ref-name>myDataSource</res-ref-name>
   <jndi-name>accountDataSource</jndi-name>
</resource-description>
```

Referencing EJBs in a Web Application

You reference EJBs in a Web Application by giving them a name in the Web Application deployment descriptor that is mapped to the JNDI name for the EJB that is defined in the `weblogic-ejb-jar.xml` file deployment descriptor.

To configure EJBs for use in a Web Application:

1. Enter the EJB reference name you use to look up the EJB in your code, the Java class name, and the class name of the home and remote interfaces of the EJB in the `<ejb-ref>` element of the Web Application deployment descriptor. For instructions on making deployment descriptor entries, see Reference EJB resources at `http://e-docs.bea.com/wls/docs60/programming/webappdeployment.html#ejb-ref`.
2. Map the reference name in `<ejb-reference-description>` element of the WebLogic-specific deployment descriptor, `weblogic.xml`, to the JNDI name defined in the `weblogic-ejb-jar.xml` file. For instructions on making deployment descriptor entries, see Map EJB resources at `http://e-docs.bea.com/wls/docs60/programming/webappdeployment.html#ejb-reference-description`.

If the Web Application is part of an Enterprise Application Archive (`.ear` file), you can reference an EJB by the name used in the `.ear` with the `<ejb-link>` element.

Setting Up Session Management

WebLogic Server is set up to handle session tracking by default. You need not set any of these properties to use session tracking. However, configuring how WebLogic Server manages sessions is a key part of tuning your application for best performance. Tuning depends upon factors such as:

- How many users you expect to hit the servlet
- How many concurrent users hit the servlet
- How long each session lasts
- How much data you expect to store for each user

HTTP Session Properties

You configure WebLogic Server session tracking with properties in the WebLogic-specific deployment descriptor, `weblogic.xml` file. For instructions on editing the WebLogic-specific deployment descriptor, see Define Session Parameters at `http://e-docs.bea.com/wls/docs60/programming/webappdeployment.html#session-descriptor`.

A complete list of session attributes is available at `http://e-docs.bea.com/wls/docs60/programming/weblogic_xml.html#session-descriptor`.

Session Timeout

You can specify an interval of time after which HTTP sessions expire. When a session expires, all data stored in the session is discarded. You can set the interval in one of two ways:

- Set the `TimeoutSecs` attribute in the `<session-descriptor>` element (see `http://e-docs.bea.com/wls/docs60/programming/weblogic_xml.html#session-descriptor`) of the WebLogic-specific deployment descriptor, `weblogic.xml`. This value is set in seconds.
- Set the `<session-timeout>` (see `http://e-docs.bea.com/wls/docs60/programming/web_xml.html#web_xml_session-config`) `element` in the Web Application deployment descriptor, `web.xml`. This value is set in minutes and overrides any value set in the `TimeoutSecs` attribute in the `<session-descriptor>` element of the WebLogic-specific deployment descriptor, `weblogic.xml`

Configuring Session Cookies

WebLogic Server uses cookies for session management when supported by the client browser.

The cookies that WebLogic Server uses to track sessions are set as transient by default and do not out-live the life of the browser. When a user quits the browser, the cookies are lost and the session lifetime is regarded as over. This behavior is in the spirit of session usage and it is recommended that you use sessions in this way.

It is possible to configure many aspects of the cookies used to track sessions with attributes that are defined in the WebLogic-specific deployment descriptor, `weblogic.xml`. A complete list of session and cookie-related attributes is available at `http://e-docs.bea.com/wls/docs60/programming/weblogic_xml.html#session-descriptor`.

For instructions on editing the WebLogic-specific deployment descriptor, see Define Session Parameters at `http://e-docs.bea.com/wls/docs60/programming/webappdeployment.html#session-descriptor`.

Using Longer-lived Cookies

For longer-lived client-side user data, your application should create and set its own cookies on the browser via the HTTP servlet API, and should not attempt to use the cookies associated with the HTTP session. Your application might use cookies to auto-login a user from a particular machine, in which case you would set a new cookie to last for a long time. Remember that the cookie can only be sent from that client machine. Your application should store data on the server if it must be accessed by the user from multiple locations.

You cannot directly connect the age of a browser cookie with the length of a session. If a cookie expires before its associated session, that session becomes orphaned. If a session expires before its associated cookie, the servlet is not be able to find a session. At that point, a new session is assigned when the `getSession()` method is called. You should only make transient use of sessions.

Configuring Session Persistence

There are four different implementations of session persistence:

- Memory (single-server, non-replicated)
- File system persistence
- JDBC persistence
- In-memory replication (across a cluster)

The first three are covered here; in-memory replication is covered in Understanding HTTP Session State Replication (at `http://e-docs.bea.com/wls/docs60/cluster/servlet.html`).

For file, JDBC, and in-memory replication, you need to set additional attributes, including `PersistentStoreType`. Each method has its own set of properties as shown below.

Common Properties

You can configure the number of sessions that are held in memory by setting the following attributes in the WebLogic-specific deployment descriptor, `weblogic.xml`. These attributes are only applicable if you are using session persistence:

`CacheSize`
: Limits the number of cached sessions that can be active in memory at any one time. If you are expecting high volumes of simultaneous active sessions, you do not want these sessions to soak up the RAM of your server since this may cause performance problems swapping to and from virtual memory. When the cache is full, the least recently used sessions are stored in the persistent store and recalled automatically when required. If you do not use persistence, this property is ignored, and there is no soft limit to the number of sessions allowed in main memory. By default, the number of cached sessions is 1024. The minimum is 16, and maximum is *`Integer.MAX_VALUE`*. An empty session uses less than 100 bytes, but grows as data is added to it.

`SwapIntervalSecs`
: The interval the server waits between purging the least recently used sessions from the cache to the persistent store, when the `cacheEntries` limit has been reached.

  If unset, this property defaults to 10 seconds; minimum is 1 second, and maximum is 604800 (1 week).

`InvalidationIntervalSecs`
: Sets the time, in seconds, that WebLogic Server waits between doing house-cleaning checks for timed-out and invalid sessions, and deleting the old sessions and freeing up memory. Set this parameter to a value less than the value set for the `<session-timeout>` element. Use this parameter to tune WebLogic Server for best performance on high traffic sites.

  The minimum value is every second (1). The maximum value is once a week (604,800 seconds). If unset, the parameter defaults to 60 seconds.

  You set `<session-timeout>` in the <session-config> element (see `http://e-docs.bea.com/wls/docs60/programming/web_xml.html#web_xml_session-config`) of the Web Application deployment descriptor `web.xml`.

Using Memory-based, Single-server, Non-replicated Persistent Storage

To use memory-based, single-server, non-replicated persistent storage, set the property `PersistentStoreType` to `memory`. When you use memory-based storage all session information is stored in memory and is lost when you stop and restart WebLogic Server.

Using File-based Persistent Storage

For file-based persistent storage for sessions:

1. Set the `PersistentStoreType` to `file`.
2. Set the directory where WebLogic Server stores the sessions. For more information, on setting this directory, see PersistentStoreDir at `http://e-docs.bea.com/wls/docs60/programming/weblogic_xml.html#PersistentStoreDir`.

   If you do not explicitly set a value for this attribute, a temporary directory is created for you by WebLogic Server.

   If you are using file-based persistence in a cluster, you must explicitly set this attribute to a shared directory that is accessible to all the servers in a cluster. You must create this directory yourself.

Using a Database for Persistent Storage

For JDBC-based persistent storage for sessions:

1. Set JDBC as the persistent store method by setting the attribute `PersistentStoreType` to `jdbc`.
2. Set a JDBC connection pool to be used for persistence storage with the `PersistentStorePool` attribute. Use the name of a connection pool that is defined in the WebLogic Server Administration Console.

For more details on setting up a database connection pool, see Managing JDBC Connectivity at `http://e-docs.bea.com/wls/docs60/adminguide/jdbc.html`.

3. Set an ACL for the connection that corresponds to the users that have permission. For more details on setting up a database connection, see Managing JDBC Connectivity at `http://e-docs.bea.com/wls/docs60/adminguide/jdbc.html`.
4. Set up a database table named `wl_servlet_sessions` for JDBC-based persistence. The connection pool that connects to the database needs to have read/write access for this table. The following table shows the Column names and data types you should use when creating this table.

Table 8-4 wl_servlet_sessions table

Column name	Type
`wl_id`	Variable-width alphanumeric column, up to 100 characters; for example, Oracle `VARCHAR2(100)`. *The primary key must be set as follows:* `wl_id + wl_context_path`.
wl_context_path	Variable-width alphanumeric column, up to 100 characters; for example, Oracle `VARCHAR2(100)`. *This column is used as part of the primary key. (See the* `wl_id` *column description.)*
`wl_is_new`	Single char column; for example, Oracle `CHAR(1)`
`wl_create_time`	Numeric column, 20 digits; for example, Oracle `NUMBER(20)`
`wl_is_valid`	Single char column; for example, Oracle `CHAR(1)`
`wl_session_values`	Large binary column; for example, Oracle `LONG RAW`
`wl_access_time`	Numeric column, 20 digits; for example, `NUMBER(20)`
wl_max_inactive_interval	Integer column; for example, Oracle `Integer`. Number of seconds between client requests before the session is invalidated. A negative time value indicates that the session should never timeout.

If you are using an Oracle DBMS, you can use the following SQL statement to create the `wl_servlet_sessions` table:

```
create table wl_servlet_sessions
  ( wl_id VARCHAR2(100) NOT NULL,
    wl_context_path VARCHAR2(100) NOT NULL,
    wl_is_new CHAR(1),
    wl_create_time NUMBER(20),
    wl_is_valid CHAR(1),
    wl_session_values LONG RAW,
    wl_access_time NUMBER(20),
    wl_max_inactive_interval INTEGER,
   PRIMARY KEY (wl_id, wl_context_path) );
```

You can modify the proceeding SQL statement for use with your DBMS.

Note: You can configure a maximum duration that the JDBC session persistence should wait for a JDBC connection from the connection pool before failing to load the session data with the `JDBCConnectionTimeoutSecs (see http://e-docs.bea.com/wls/docs60/programming/weblogic_xml.html#JDBCConnectionTimeoutSecs)` attribute.

Using URL Rewriting

In some situations, a browser may not accept cookies, which makes session tracking using cookies impossible. URL rewriting is a solution to this situation that can be substituted automatically when WebLogic Server detects that the browser does not accept cookies. URL rewriting involves encoding the session ID into the hyper-links on the Web pages that your servlet sends back to the browser. When the user subsequently clicks these links, WebLogic Server extracts the ID from the URL address and finds the appropriate `HttpSession` when your servlet calls the `getSession()` method.

To enable URL rewriting in WebLogic Server, set the `URLRewritingEnabled (see http://e-docs.bea.com/wls/docs60/programming/weblogic_xml.html#URLRewritingEnabled)` attribute in the WebLogic-specific deployment descriptor, `weblogic.xml`, under the `<session-descriptor>` element. to `true`. (The default value for this attribute is `true`.)

Coding Guidelines for URL Rewriting

There are some general guidelines for how your code should handle URLs in order to support URL rewriting.

- Avoid writing a URL straight to the output stream, as shown here:

  ```
  out.println("<a href=\"/myshop/catalog.jsp\">catalog</a>");
  ```

  Instead, use the `HttpServletResponse.encodeURL()` method, for example:

  ```
  out.println("<a href=\""
       + response.encodeURL("myshop/catalog.jsp")
       + "\">catalog</a>");
  ```

  Calling the `encodeURL()` method determines if the URL needs to be rewritten, and if so, it rewrites it, by including the session ID in the URL. The session ID is appended to the URL and begins with a semicolon.

- In addition to URLs that are returned as a response to WebLogic Server, also encode URLs that send redirects. For example:

  ```
  if (session.isNew())
    response.sendRedirect
  (response.encodeRedirectUrl(welcomeURL));
  ```

  WebLogic Server uses URL rewriting when a session is new, even if the browser does accept cookies, because the server cannot tell if a browser accepts cookies in the first visit of a session.

- Your servlet can determine if a given session ID was received from a cookie by checking the Boolean returned from the `HttpServletRequest.isRequestedSessionIdFromCookie()` method. Your application may respond appropriately, or simply rely on URL rewriting by WebLogic Server.

URL Rewriting and Wireless Access Protocol (WAP)

If you are writing a WAP application, you must use URL rewriting because the WAP protocol does not support cookies. In addition, some WAP devices have a 128-character limit on the length of a URL (including parameters), which limits the amount of data that can be transmitted using URL rewriting. To allow more space for parameters, you can limit the size of the session ID that is randomly generated by

WebLogic Server by specifying the number of bytes with the IDLength (see http://e-docs.bea.com/wls/docs60/programming/weblogic_xml.html#IDLength) attribute.

Using Character Sets and POST Data

You can set the character set that is used when processing data from a form that uses the POST method. To inform the application that processes the form data that a particular character set is in use, you add specific "signal" characters to the URL used to process the form data (specified with the action attribute of the <form> tag) and then map those characters to an encoding in the Web Application deployment descriptor, web.xml. POST data is normally read as ASCII unless specified using the following procedure.

To process POST data in a non-ASCII character set:

1. Create an entry in the Web Application deployment descriptor, web.xml, within a <context-param> element. This entry should come after the <distributable> element and before the <servlet> element in the web.xml file. In this entry, the <param-name> always includes the class name weblogic.httpd.inputCharset, followed by a period, followed by the signal string. The <param-value> contains the name of the HTTP character set. In the following example, the string /rus/jo* is mapped to the windows-1251 character set:

   ```
   <context-param>
     <param-name>weblogic.httpd.inputCharset./rus/jo*</param-name>
     <param-value>windows-1251</param-value>
   </context-param>
   ```

2. Code the HTML form to use the signal string when sending the form data. For example:

   ```
   <form action="http://some.host.com/myWebApp/rus/jo/index.html">
     ...
   </form>
   ```

   Place the signal string *after* the Web Application name (also called the context path—myWebApp— in this case) and *before* the remaining portion of the URL.

For more information on the Web Application deployment descriptor, see Define Context Parameters at `http://e-docs.bea.com/wls/docs60/programming/webappdeployment.html#context-param`.

9 Installing and Configuring the Apache HTTP Server Plug-In

The following sections describe how to install and configure the Apache HTTP Server Plug-In:

Overview

The Apache HTTP Server Plug-In allows requests to be proxied from an Apache HTTP Server to WebLogic Server. The plug-in enhances an Apache installation by allowing WebLogic Server to handle those requests that require the dynamic functionality of WebLogic Server.

The plug-in is intended for use in an environment where an Apache Server serves static pages, and another part of the document tree (dynamic pages best generated by HTTP Servlets or JavaServer Pages) is delegated to WebLogic Server, which may be operating in a different process, possibly on a different host. To the end user — the browser — the HTTP requests delegated to WebLogic Server still appear to be coming from the same source.

The HTTP-tunneling can also operate through the plug-in, providing non-browser clients access to WebLogic Server services.

The Apache HTTP Server Plug-In operates as an Apache module within an Apache HTTP Server. An Apache module is loaded by Apache Server at startup, and then certain HTTP requests are delegated to it. Apache modules are similar to HTTP servlets, except that an Apache module is written in code native to the platform.

Keep-Alive Connections in Apache

The Apache HTTP Server Plug-In creates a socket for each request and closes the socket after reading the response. Because Apache HTTP Server is multiprocessed, connection pooling and keep-alive connections between WebLogic Server and theApache HTTP Server Plug-In cannot be supported.

Proxying Requests

The plug-in proxies requests to WebLogic Server based on a configuration that you specify. You can proxy requests either based on the URL of the request (or a portion of the URL). This is called proxying by *path*. You can also proxy request based on the *MIME type* of the requested file. You can also use a combination of both methods. If

a request matches both criteria, the request is proxied by path. You can also specify additional parameters for each of these types of requests that define additional behavior of the plug-in. For more information, see "Configuring the Apache HTTP Server Plug-In" on page 9-6.

Platform Support

The Apache HTTP Server Plug-In is supported on Linux, Solaris and HPUX11 platforms. For information on support for specific versions of Apache, see the BEA WebLogic Server Platform Support Page at http://e-docs.bea.com/wls/platforms/index.html#apach.

Installing the Apache HTTP Server Plug-In

You install the Apache HTTP Server Plug-In as an Apache module along with your Apache HTTP Server installation. To install the Apache HTTP Server Plug-In:

1. Locate the shared object file for your platform.

   The Apache plug-in is distributed as a shared object (`.so`) for use on Solaris, Linux, and HPUX11 platforms. Each shared object file is distributed as separate versions, depending on the platform, whether or not SSL is to be used between the client and Apache, and the encryption strength for SSL (regular or 128 bit). The shared object files are located in the following directories of your WebLogic Server installation:

   Solaris
   `lib/solaris`

   Linux
   `lib/linux`

   HPUX11
   `lib/hpux11`

Choose the appropriate shared object from the following table.:

Apache Version	Regular Strength Encryption	128-bit Encryption
Standard Apache, Version 1.x	`mod_wl.so`	`mod_wl128.so`
Apache w/ SSL/EAPI Version 1.x (Stronghold, modssl etc).	`mod_wl_ssl.so`	`mod_wl128_ssl.so`
Apache + Raven Version 1.x Required because Raven applies frontpage patches that makes the plug-in incompatible with the standard shared object.	`mod_wl_ssl_raven.so`	`mod_wl128_ssl_raven.so`

2. Enable the shared object.

   The Apache HTTP Server Plug-In will be installed as an Apache Dynamic Shared Object (DSO). DSO support in Apache is based on a module named `mod_so.c` that must be enabled before `mod_wl.so` is loaded. If you installed Apache using the supplied script, `mod_so.c` should already be enabled. To verify that `mod_so.c` is enabled, execute the following command:

   `APACHE_HOME``/bin/httpd -l`

   (Where *`APACHE_HOME`* is the directory containing your Apache HTTP Server installation.)

   This command lists all of the enabled modules. If `mod_so.c` is not listed, build your Apache HTTP Server from the source code, making sure that the following options are configured:

```
...
--enable-module=so
--enable-rule=SHARED_CORE
...
```

3. Install the Apache HTTP Server Plug-In with a support program called `apxs` (APache eXtenSion) that builds DSO-based modules outside of the Apache source tree, and adds the following line to the `httpd.conf` file:

   ```
   AddModule mod_so.c
   ```

   In your WebLogic Server installation, use a command shell to navigate to the directory that contains the shared object for your platform and activate the `weblogic_module` by issuing this command (note that you must have Perl installed to run this Perl script):

   ```
   perl APACHE_HOME/bin/apxs –i –a –n weblogic mod_wl.so
   ```

   This command copies the `mod_wl.so` file to the *`APACHE_HOME`*`/libexec` directory. It also adds two lines of instructions for `weblogic_module` to the `httpd.conf` file and activates the module. Make sure that the following lines were added to your *`APACHE_HOME`*`/conf/httpd.conf` file:

   ```
   LoadModule weblogic_module
   AddModule mod_weblogic.c
   ```

4. Verify the syntax of the *`APACHE_HOME`*`/conf/httpd.conf` file with the following command:

   ```
   APACHE_HOME/bin/apachect1 configtest
   ```

   The output of this command indicates any errors in your `httpd.conf` file.

5. Configure any additional parameters in the Apache `httpd.conf` configuration file as described in the section "Configuring the Apache HTTP Server Plug-In" on page 9-6. The `httpd.conf` file allows you to customize the behavior of the Apache HTTP Server Plug-In.

6. Start Weblogic Server.

7. Start (or restart if you have changed the configuration) Apache HTTP Server.

8. Test the Apache plug-in by opening a browser and setting the URL to the Apache Server + "`/weblogic/`", which should bring up the default WebLogic Server HTML page, welcome file, or default servlet, as defined for the default Web Application on WebLogic Server. For example:

   ```
   http://myApacheserver.com/weblogic/
   ```

Configuring the Apache HTTP Server Plug-In

After you install the plug-in (see "Installing the Apache HTTP Server Plug-In" on page 9-3), edit the `httpd.conf` file to configure the Apache plug-in. Editing the `httpd.conf` file informs the Apache web server that it should load the native library for the plug-in as an Apache module and also describes which requests should be handled by the module.

Editing the httpd.conf File

To edit the `httpd.conf` file to configure the Apache HTTP Server Plug-In:

1. Open the `httpd.conf` file. The file is located at *`APACHE_HOME`*`/conf/httpd.conf` (where *`APACHE_HOME`* is the root directory of your Apache installation).
2. Verify that the following two lines were added to the `httpd.conf` file when you ran the `apxs` utility:

   ```
   LoadModule weblogic_module    libexec/mod_wl.so
   AddModule mod_weblogic.c
   ```

3. Add an `IfModule` block that defines one of the following:

   For a *non-clustered* WebLogic Server:
   The `WebLogicHost` and `WebLogicPort` parameters.

   For a *cluster* of WebLogic Servers:
   The `WebLogicCluster` parameter.

   For example:

   ```
   <IfModule mod_weblogic.c>
     WebLogicHost myweblogic.server.com
     WebLogicPort 7001
   </IfModule>
   ```

4. If you want to proxy requests by MIME type, also add a `MatchExpression` line to the `IfModule` block. (You can also proxy requests by path. Proxying by path takes precedence over proxying by MIME type. If you only want to proxy requests by path, skip to step)

For example, the following `IfModule` block for a non-clustered WebLogic Server specifies that all files with MIME type `.jsp` are proxied:

```
<IfModule mod_weblogic.c>
  WebLogicHost myweblogic.server.com
  WebLogicPort 7001
  MatchExpression *.jsp
</IfModule>
```

You can also use multiple `MatchExpressions`, for example:

```
<IfModule mod_weblogic.c>
  WebLogicHost myweblogic.server.com
  WebLogicPort 7001
  MatchExpression *.jsp
  MatchExpression *.xyz
</IfModule>
```

If you are proxying requests by MIME type to a cluster of WebLogic Servers, use the `WebLogicCluster` parameter instead of the `WebLogicHost` and `WebLogicPort` parameters. For example:

```
<IfModule mod_weblogic.c>
  WebLogicCluster w1s1.com:7001,w1s2.com:7001,w1s3.com:7001
  MatchExpression *.jsp
  MatchExpression *.xyz
</IfModule>
```

5. If you want to proxy requests by path, use the `Location` block and the `SetHandler` statement. `SetHandler` specifies the handler for the Apache HTTP Server Plug-In module. For example the following Location block proxies all requests containing the `/weblogic` in the URL:

   ```
   <Location /weblogic>
     SetHandler weblogic-handler
   </Location>
   ```

6. Define any additional parameters for the Apache HTTP Server Plug-In.

   The Apache HTTP Server Plug-In recognizes the parameters listed in "General Parameters for Web Server Plug-Ins" on page C-2. To modify the behavior of your Apache HTTP Server Plug-In, define these parameters either:

   - in a `Location` block, for parameters that apply to proxying by *path*, or
   - in an `IfModule` block, for parameters that apply to proxying by *MIME type*.

Notes on Editing the httpd.conf File

- As an alternative to the procedure in "Editing the httpd.conf File" on page 9-6, You can define parameters in a separate file called `weblogic.conf` file that is *included* in the `IfModule` block. Using this included file may help modularize your configuration.For example

  ```
  <IfModule mod_weblogic.c>
    # Config file for WebLogic Server that defines the parameters
    Include conf/weblogic.conf
  </IfModule>
  ```

  Note: Defining parameters in an *included* file is not supported when using SSL between Apache HTTP Server Plug-In and WebLogic Server.

- Each parameter should be entered on a new line. Do not put an '=' between the parameter and its value. For example:

  ```
  PARAM_1 value1
  PARAM_2 value2
  PARAM_3 value3
  ```

- If a request matches both a MIME type specified in a `MatchExpression` in an `IfModule` block and a path specified in a `Location` block, the behavior specified by the `Location` block takes precedence.

- If you define the `CookieName` parameter, you must define it in an `IfModule` block.

Using SSL With the Apache Plug-In

You can use the Secure Sockets Layer (SSL) protocol to protect the connection between the Apache HTTP Server Plug-In and WebLogic Server. The SSL protocol provides confidentiality and integrity to the data passed between the Apache HTTP Server Plug-In and WebLogic Server. In addition, the SSL protocol allows the plug-in to authenticate itself to WebLogic Server to ensure that information is passed to a trusted principal.

The Apache HTTP Server Plug-In does *not* use the transport protocol (`http` or `https`) specified in the HTTP request (usually by the browser) to determine whether or not the SSL protocol is used to protect the connection between the Apache HTTP Server Plug-In and WebLogic Server.

Note: You cannot configure a 2-way SSL between the Apache HTTP Server and WebLogic Server. The SSL protocol is a point-to-point connection, cyptographically sealed end-to-end. Therefore, any type of proxy or firewall cannot see into the SSL socket. The Apache HTTP Server acts as the server end-point in the SSL connection. The configuration is:

```
client-->2-way SSL-->Apache<--1-way SSL<--WebLogic Server
```

The Apache HTTP Server cannot use the digital certificate from the first SSL connection in the second SSL connection because it cannot use the client's private key.

Configuring SSL Between the Apache HTTP Server Plug-In and WebLogic Server

To use the SSL protocol between Apache HTTP Server Plug-In and WebLogic Server:

1. Configure WebLogic Server for SSL. For more information, see "Configuring the SSL Protocol" on page 12-28.
2. Configure the WebLogic Server SSL listen port. For more information, see "Configuring the Listen Port" on page 7-3.
3. Set the `WebLogicPort` parameter in the `httpd.conf` file to the listen port configured in step 2.
4. Set the `SecureProxy` parameter in the `httpd.conf` file to `ON`.
5. Set any additional parameters in the `httpd.conf` file that define information about the SSL connection. For a complete list of parameters, see "SSL Parameters for Web Server Plug-Ins" on page C-11.

Issues with SSL-Apache Configuration

Two known issues arise when you configure the Apache plug-in to use SSL:

- The `PathTrim` (see page C-3) parameter must be configured inside the `<Location>` tag.

  The following configuration is **incorrect**:

  ```
  <Location /weblogic>
   SetHandler weblogic-handler
  </Location>
  ```

  ```
  <IfModule mod_weblogic.c>
   WebLogicHost localhost
   WebLogicPort 7001
   PathTrim /weblogic
  </IfModule>
  ```

  The following configuration is the **correct** setup:

  ```
  <Location /weblogic>
    SetHandler weblogic-handler
    PathTrim /weblogic
  </Location>
  ```

- The `Include` directive does not work with Apache SSL. You must configure all parameters directly in the `httpd.conf` file. Do not use the following configuration when using SSL:

  ```
  <IfModule mod_weblogic.c>
    MatchExpression *.jsp
    Include weblogic.conf
  </IfModule>
  ```

Connection Errors and Clustering Failover

When the Apache HTTP Server Plug-In attempts to connect to WebLogic Server, the plug-in uses several configuration parameters to determine how long to wait for connections to the WebLogic Server host and, after a connection is established, how long the plug-in waits for a response. If the plug-in cannot connect or does not receive a response, the plug-in will attempt to connect and send the request to other WebLogic Servers in the cluster. If the connection fails or there is no response from any WebLogic Server in the cluster, an error message is sent.

Figure 9-1 “Connection Failover” on page 9-13 demonstrates how the plug-in handles failover.

Connection Failures

Failure of the host to respond to a connection request could indicate possible problems with the host machine, networking problems, or other server failures.

Failure of WebLogic Server to respond, could indicate that WebLogic Server is not running or is unavailable, a hung server, a database problem, or other application failure.

Failover with a Single, Non-Clustered WebLogic Server

If you are running only a single WebLogic Server the same logic described here applies, except that the plug-in only attempts to connect to the server defined with the `WebLogicHost` parameter. If the attempt fails, an `HTTP 503` error message is returned. The plug-in continues trying to connect to WebLogic Server until `ConnectTimeoutSecs` is exceeded.

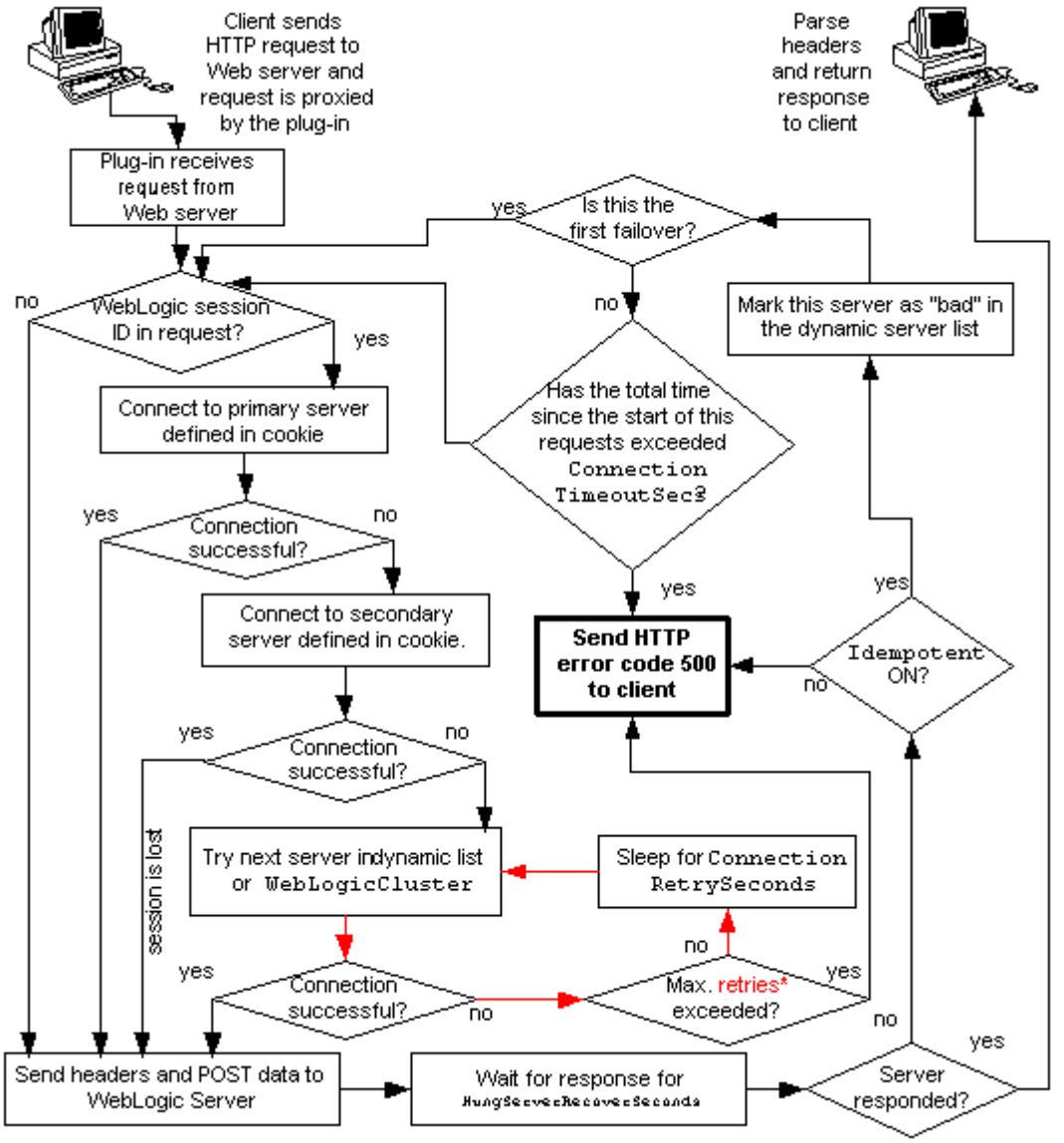
The Dynamic Server List

When you specify a list of WebLogic Servers in the `WebLogicCluster` parameter, the plug-in uses that list as a starting point for load balancing among the members of the cluster. After the first request is routed to one of these servers, a dynamic server list is returned containing an updated list of servers in the cluster. The updated list adds any new servers in the cluster and deletes any that are no longer part of the cluster or that have failed to respond to requests. This list is updated automatically with the HTTP response when a change in the cluster occurs.

Failover, Cookies, and HTTP Sessions

When a request contains a session information stored in a cookie, in the POST data, or by URL encoding, the session ID contains a reference to the specific server in which the session was originally established (called the *primary* server) and a reference to an additional server where the original session is replicated (called the *secondary* server). A request containing a cookie attempts to connect to the primary server If that attempt fails, the request is routed to the secondary server. If both the primary and secondary servers fail, the session is lost and the plug-in attempts to make a fresh connection to another server in the dynamic cluster list. For more information see Figure 9-1 "Connection Failover" on page 9-13.

Figure 9-1 Connection Failover



*The Maximum number of retries allowed in the red loop is equal to `ConnectTimeoutSecs ÷ ConnectRetrySecs.`

Template for the httpd.conf File

This section contains a sample `httpd.conf` file. You can use this sample as a template that you can modify to suit your environment and server. Lines beginning with # are comments. Note that Apache HTTP Server is not case sensitive, and that the `LoadModule` and `AddModule` lines are automatically added by the `apxs` utility.

```
########################################################

APACHE-HOME/conf/httpd.conf file
########################################################

LoadModule weblogic_module   libexec/mod_wl.so

AddModule mod_weblogic.c

<Location /weblogic>
  SetHandler weblogic-handler
  PathTrim /weblogic
  ErrorPage http://myerrorpage1.mydomain.com
</Location>

<Location /servletimages>
  SetHandler weblogic-handler
  PathTrim /something
  ErrorPage http://myerrorpage1.mydomain.com
</Location>

<IfModule mod_weblogic.c>
  MatchExpression *.jsp
  WebLogicCluster w1s1.com:7001,w1s2.com:7001,w1s3.com:7001
  ErrorPage http://myerrorpage.mydomain.com
</IfModule>
```

Sample Configuration Files

Instead of defining parameters in the `location` block of your `httpd.conf` file, if you prefer, you can use a `weblogic.conf` file that is loaded by the `IfModule` in the `httpd.conf` file. The following examples may be used as templates that you can modify to suit your environment and server. Lines beginning with '#' are comments.

Example Using WebLogic Clusters

```
# These parameters are common for all URLs which are
# directed to the current module. If you want to override
# these parameters for each URL, you can set them again in
# the <Location> or <Files> blocks. (Except WebLogicHost,
# WebLogicPort, WebLogicCluster, and CookieName.)

<IfModule mod_weblogic.c>
  WebLogicCluster w1s1.com:7001,w1s2.com:7001,w1s3.com:7001
  ErrorPage http://myerrorpage.mydomain.com
  MatchExpression *.jsp
</IfModule>
#####################################################
```

Example Using Multiple WebLogic Clusters

```
# These parameters are common for all URLs which are
# directed to the current module. If you want to override
# these parameters for each URL, you can set them again in
# the <Location> or <Files> blocks (Except WebLogicHost,
# WebLogicPort, WebLogicCluster, and CookieName.)

<IfModule mod_weblogic.c>
  MatchExpression *.jsp WebLogicHost=myHost|WebLogicPort=7001|Debug=ON
  MatchExpression *.html WebLogicCluster=myHost1:7282,myHost2:7283|ErrorPage=
    http://www.xyz.com/error.html
</IfModule>
```

Example Without WebLogic Clusters

```
# These parameters are common for all URLs which are
# directed to the current module. If you want to override
# these parameters for each URL, you can set them again in
# the <Location> or <Files> blocks (Except WebLogicHost,
# WebLogicPort, WebLogicCluster, and CookieName.)

<IfModule mod_weblogic.c>
  WebLogicHost myweblogic.server.com
  WebLogicPort 7001
  MatchExpression *.jsp
</IfModule>
```

Example Configuring IP-Based Virtual Hosting

```
NameVirtualHost 172.17.8.1
<VirtualHost goldengate.domain1.com>
WebLogicCluster tehama1:4736,tehama2:4736,tehama:4736
PathTrim /x1
ConnectTimeoutSecs 30
</VirtualHost>
<VirtualHost goldengate.domain2.com>
WeblogicCluster green1:4736,green2:4736,green3:4736
PathTrim /y1
ConnectTimeoutSecs 20
</VirtualHost>
```

Example Configuring Name-Based Virtual Hosting With a Single IP Address

```
<VirtualHost 162.99.55.208>
        ServerName myserver.mydomain.com
         <Location / >
               SetHandler weblogic-handler
               WebLogicCluster 162.99.55.71:7001,162.99.55.72:7001
               Idempotent ON
               Debug ON
               DebugConfigInfo ON
        </Location>
</VirtualHost>

<VirtualHost 162.99.55.208>
        ServerName myserver.mydomain.com
       <Location / >
                SetHandler weblogic-handler
                WebLogicHost russell
                WebLogicPort 7001
                Debug ON
                DebugConfigInfo ON
        </Location>
</VirtualHost>
```

CHAPTER

10 Installing and Configuring the Microsoft Internet Information Server (ISAPI) Plug-In

This following sections describes how to install and configure the Microsoft Internet Information Server Plug-In. The following topics are covered:

Overview of the Microsoft Internet Information Server Plug-In

The Microsoft Internet Information Server Plug-In allows requests to be proxied from a Microsoft Internet Information Server (IIS) to WebLogic Server. The plug-in enhances an IIS installation by allowing WebLogic Server to handle those requests that require the dynamic functionality of WebLogic Server.

The Microsoft Internet Information Server Plug-In is intended for use in an environment where the Internet Information Server (IIS) serves static pages such as HTML pages, while dynamic pages such as HTTP Servlets or JavaServer Pages are served by WebLogic Server. The WebLogic Server may be operating in a different process, possibly on a different host. To the end user—the browser—the HTTP requests delegated to WebLogic Server still appear to be coming from IIS. The HTTP-tunneling facility of the WebLogic client-server protocol also operates through the plug-in, providing access to all WebLogic Server services.

Connection Pooling and Keep-Alive

The Microsoft Internet Information Server Plug-In improves performance by using a re-usable pool of connections from the plug-in to WebLogic Server. The plug-in implements HTTP 1.1 keep-alive connections between the plug-in and WebLogic Server by re-using the same connection in the pool for subsequent requests from the same client. If the connection is inactive for more than 30 seconds, (or a user-defined amount of time) the connection is closed and returned to the pool.

Proxying Requests

The plug-in proxies requests to WebLogic Server based on a configuration that you specify. You can proxy requests either based on the URL of the request (or a portion of the URL). This is called proxying by *path*. You can also proxy request based on the *MIME type* of the requested file. You can also use a combination of both methods. If a request matches both criteria, the request is proxied by path. You can also specify

additional parameters for each of these types of requests that define additional behavior of the plug-in. For more information, see "Installing the Microsoft Internet Information Server Plug-In" on page 10-3.

Platform Support

For the latest information on operating system and IIS version compatibility with the Microsoft Internet Information Server Plug-In, see the platform support page at `http://e-docs.bea.com/wls/platforms/index.html#iis`.

Installing the Microsoft Internet Information Server Plug-In

To install the Microsoft Internet Information Server Plug-In:

1. Copy the `iisproxy.dll` file from the `/bin` directory of your WebLogic Server installation into a convenient directory that is accessible by IIS. This directory must also contain the `iisproxy.ini` file.
2. Start the IIS Internet Service Manager by selecting it from the Microsoft IIS Start menu.
3. In the left panel of the Service Manager, select your website (the default is "Default Web Site").
4. Click the "Play" arrow in the toolbar to start.
5. Open the properties for the selected website by holding the right mouse button down over the website selection in the left panel.
6. In the Properties panel, select the Home Directory tab, and click the Configuration button in the Applications Settings section.
7. Configure proxying by file type:

a. On the App Mappings tab, click the Add button to add file types and configure them to be proxied to WebLogic Server.

b. In the dialog box, browse to find the "`iisproxy.dll`" file.

c. Set the Extension to the type of file that you want to proxy to WebLogic Server.

d. Select the "Script engine" check box.

e. Set Execute Permissions to "Scripts and Executables".

f. Deselect the "Check that file exists" check box.

g. Set the Method exclusions as needed to create a secure installation.

h. When you finish, click the OK button to save the configuration. Repeat this process for each file type you want to proxy to WebLogic.

i. When you finish configuring file types, click the OK button to close the Properties panel.

Note: Any path information you add to the URL after the server and port is passed directly to WebLogic Server. For example, if you request a file from IIS with the URL:

```
http://myiis.com/jspfiles/myfile.jsp
```

it is proxied to WebLogic Server with a URL such as

```
http://mywebLogic:7001/jspfiles/myfile.jsp
```

8. Create the `iisproxy.ini` file.

   The `iisproxy.ini` file contains name=value pairs that define configuration parameters for the plug-in. The parameters are listed in "General Parameters for Web Server Plug-Ins" on page C-2.

   Note: Changes in the parameters will not go into effect until you restart the "IIS Admin Service" (Under *services*, in the control panel).

   BEA recommends that you locate the `iisproxy.ini` file in the same directory that contains the `iisproxy.dll` file. You can also use other locations. If you place the file elsewhere, note that WebLogic Server searches for `iisproxy.ini` in the following directories, in the following order:

   a. The same directory where `iisproxy.dll` is located.

b. The home directory of the most recent version of WebLogic Server that is referenced in the Windows Registry. If WebLogic Server does not find the `iisproxy.ini` file there, it continues looking in the Windows Registry for older versions of WebLogic Server and looks for the `iisproxy.ini` file in the home directories of those installations.

c. `c:\weblogic`

9. Define the WebLogic Server host and port number to which the Microsoft Internet Information Server Plug-In proxies requests. Depending on your configuration, there are two ways to define the host and port:

   - If you are proxying requests to a single WebLogic Server, define the `WebLogicHost` and `WebLogicPort` parameters in the `iisproxy.ini` file. For example:

     ```
     WebLogicHost=localhost
     WebLogicPort=7001
     ```

   - If you are proxying requests to a cluster of WebLogic Servers, define the `WebLogicCluster` parameter in the `iisproxy.ini` file. For example:

     ```
     WebLogicCluster=myweblogic.com:7001,yourweblogic.com:7001
     ```

10. Configure proxying by path. In addition to proxying by file type you can configure the Microsoft Internet Information Server Plug-In to serve files based on their *path* by specifying some additional parameters in the `iisproxy.ini` file. Proxying by path takes precedence over proxying by MIME type.

    To configure proxying by path:

    a. Place the `iisforward.dll` file in the same directory as the `iisproxy.dll` file and add the `iisforward.dll` file as a filter service in IIS (WebSite *Properties* → ISAPI Filters tab → Add the `iisforward dll`).

    b. Register `.wlforward` as a special file type to be handled by `iisproxy.dll`.

    c. Define the property `WlForwardPath` in `iisproxy.ini`. `WlForwardPath` defines the path that is proxied to WebLogic Server, for example: `WlForwardPath=/weblogic`.

    d. Set the `PathTrim` parameter to trim off the `WlForwardPath` when necessary. For example, using

       ```
       WlForwardPath=/weblogic
       PathTrim=/weblogic
       ```

trims a request from IIS to Weblogic Server. Therefore, `/weblogic/session` is changed to `/session`.

e. If you want requests that do not contain extra path information (in other words, requests containing only a host name), set the `DefaultFileName` parameter to the name of the welcome page of the Web Application to which the request is being proxied. The value of this parameter is appended to the URL.

f. If you need to debug your application, set the `Debug=ON`" parameter in `iisproxy.ini`. A `c:\tmp\iisforward.log` is generated containing a log of the plug-in's activity that you can use for debugging purposes.

11. Set any additional parameters in the `iisproxy.ini` file. A complete list of parameters is available in the appendix "General Parameters for Web Server Plug-Ins" on page C-2.

12. If you are proxying servlets from IIS to WebLogic Server and you are not proxying by path, please read the section "Proxying Servlets From IIS to WebLogic Server" on page 10-9.

Creating ACLs through IIS

ACLs will not work through the Microsoft Internet Information Server Plug-In if the Authorization header is not passed by IIS. Use the following information to ensure that the Authorization header is passed by IIS.

When using Basic Authentication, the user is logged on with local log-on rights. To enable the use of Basic Authentication, grant each user account the *Log On Locally* user right on the IIS server. Note the following two problems that may result from Basic Authentication's use of local logon.

- If the user does not have local log-on rights, Basic Authentication will not work even if the FrontPage, IIS, and Windows NT configurations appear to be correct.
- A user who has local log-on rights and who can obtain physical access to the host computer running IIS will be permitted to start an interactive session at the console.

To enable Basic Authentication, in the Directory Security tab of the console, ensure that the Allow Anonymous option is "on" and all other options are "off".

Sample iisproxy.ini File

Here is a sample `iisproxy.ini` file for use with a single, non-clustered WebLogic Server. Comment lines are denoted with the "#" character.

```
# This file contains initialization name/value pairs
# for the IIS/WebLogic plug-in.

WebLogicHost=localhost
WebLogicPort=7001
ConnectTimeoutSecs=20
ConnectRetrySecs=2
```

Here is a sample `iisproxy.ini` file with clustered WebLogic Servers. Comment lines are denoted with the "#" character.

```
# This file contains initialization name/value pairs
# for the IIS/WebLogic plug-in.

WebLogicCluster=myweblogic.com:7001,yourweblogic.com:7001
ConnectTimeoutSecs=20
ConnectRetrySecs=2
```

Note: If you are using SSL between the plug-in and WebLogic Server the port number should be defined as the SSL listen port.

Using SSL with the Microsoft Internet Information Server Plug-In

You can use the Secure Sockets Layer (SSL) protocol to protect the connection between the WebLogic Server proxy plug-in and the Microsoft Internet Information Server. The SSL protocol provides confidentiality and integrity to the data passed between the Microsoft Internet Information Server Plug-In and WebLogic Server. In addition, the SSL protocol allows the WebLogic Server proxy plug-in to authenticate itself to the Microsoft Internet Information Server to ensure that information is passed to a trusted principal.

The Microsoft Internet Information Server Plug-In does not use the transport protocol (`http` or `https`) to determine whether or not the SSL protocol will be used to protect the connection between the proxy plug-in and the Microsoft Internet Information Server. In order to use the SSL protocol with the Microsoft Internet Information Server Plug-In, configure the WebLogic Server receiving the proxied requests to use the SSL protocol. The port on the WebLogic Server that is configured for secure SSL communication is used by the WebLogic Server proxy plug-in to communicate with the Microsoft Internet Information Server.

Note: You cannot configure a 2-way SSL between the Microsoft Internet Information Server and WebLogic Server. The SSL protocol is a point-to-point connection, cyptographically sealed end-to-end. Therefore, any type of proxy or firewall cannot see into the SSL socket. The Microsoft Internet Information Server acts as the server end-point in the SSL connection. The configuration is:

```
client-->2-way SSL-->IIS<--1-way SSL<--WebLogic Server
```

The Microsoft Internet Information Server cannot use the digital certificate from the first SSL connection in the second SSL connection because it cannot use the client's private key.

To use the SSL protocol between Microsoft Internet Information Server Plug-In and WebLogic Server:

1. Configure WebLogic Server for SSL. For more information, see "Configuring the SSL Protocol" on page 12-28.
2. Configure the WebLogic Server SSL listen port. For more information, see "Configuring the Listen Port" on page 7-3.
3. Set the `WebLogicPort` parameter in the `iisproxy.ini` file to the listen port configured in step 2.
4. Set the `SecureProxy` parameter in the `iisproxy.ini` file to `ON`.
5. Set additional parameters in the `iisproxy.ini` file that define the SSL connection. For a complete list of parameters, see "SSL Parameters for Web Server Plug-Ins" on page C-11.

For example:

```
WebLogicHost=myweblogic.com
WebLogicPort=7002
SecureProxy=ON
```

Proxying Servlets From IIS to WebLogic Server

Servlets may be proxied by path if the `iisforward.dll` is registered as a filter. You would then invoke your servlet with a URL similar to the following:

```
http://weblogic:7001/weblogic/myServlet
```

To proxy servlets if `iisforward.dll` is not registered as a filter, you must configure proxying by file type.To proxy servlets by file type:

1. Register an arbitrary file type (extension) with IIS to proxy the request to the WebLogic Server, as described on page 10-3. in step 7. under "Installing the Microsoft Internet Information Server Plug-In".
2. Register your servlet in the appropriate Web Application. For more information on registering servlets, see Configuring Servlets at `http://e-docs.bea.com/wls/docs60/programming.html#configuring-servlets`.
3. Invoke your servlet with a URL formed according to this pattern:

   `http://www.myserver.com/`*`virtualName/anyfile.ext`*

   where *`virtualName`* is the URL pattern defined in the `<servlet-mapping>` element of the Web Application deployment descriptor (`web.xml`) for this servlet. and *`ext`* is a file type (extension) registered with IIS for proxying to WebLogic Server. The *`anyfile`* part of the URL is ignored in this context.

   Note:

   - If the image links called from the servlet are part of the Web Application, you must also proxy the requests for the images to WebLogic Server by registering the appropriate file types (probably `.gif` and `.jpg`) with IIS. You can, however, choose to serve these images directly from IIS if desired.
   - If the servlet being proxied has links that call other servlets, then these links must also be proxied to WebLogic Server, conforming to the pattern shown above.

Testing the Installation

After you install and configure the Microsoft Internet Information Server Plug-In, follow these steps for deployment and testing:

1. Make sure WebLogic Server and IIS are running.
2. Save a JSP file into the document root of the default Web Application.
3. Open a browser and set the URL to the IIS + `filename.jsp` as shown in this example:

   ```
   http://myii.server.com/filename.jsp
   ```

   If `filename.jsp` is displayed in your browser, the plug-in is functioning.

Connection Errors and Clustering Failover

When the Microsoft Internet Information Server Plug-In attempts to connect to WebLogic Server, the plug-in uses several configuration parameters to determine how long to wait for connections to the WebLogic Server host and, after a connection is established, how long the plug-in waits for a response. If the plug-in cannot connect or does not receive a response, the plug-in will attempt to connect and send the request to other WebLogic Servers in the cluster. If the connection fails or there is no response from any WebLogic Server in the cluster, an error message is sent.

Figure 10-1 “Connection Failover” on page 10-13 demonstrates how the plug-in handles failover.

Connection Failures

Failure of the host to respond to a connection request could indicate possible problems with the host machine, networking problems, or other server failures.

Failure of WebLogic Server to respond, could indicate that WebLogic Server is not running or is unavailable, a hung server, a database problem, or other application failure.

Failover with a Single, Non-Clustered WebLogic Server

If you are running only a single WebLogic Server the same logic described here applies, except that the plug-in only attempts to connect to the server defined with the `WebLogicHost` parameter. If the attempt fails, an `HTTP 503` error message is returned. The plug-in continues trying to connect to WebLogic Server until `ConnectTimeoutSecs` is exceeded.

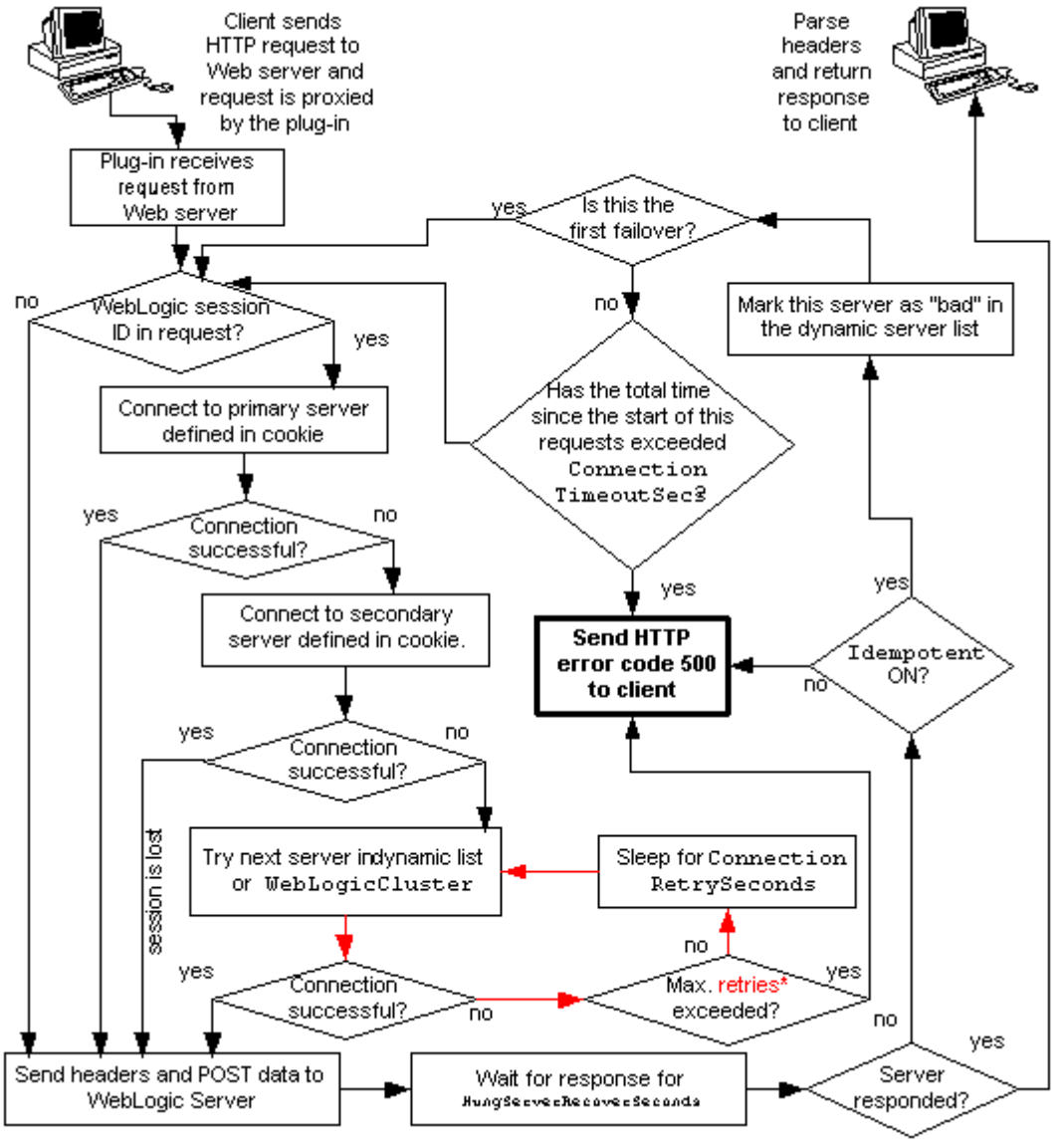
The Dynamic Server List

When you specify a list of WebLogic Servers in the `WebLogicCluster` parameter, the plug-in uses that list as a starting point for load balancing among the members of the cluster. After the first request is routed to one of these servers, a dynamic server list is returned containing an updated list of servers in the cluster. The updated list adds any new servers in the cluster and deletes any that are no longer part of the cluster or that have failed to respond to requests. This list is updated automatically with the HTTP response when a change in the cluster occurs.

Failover, Cookies, and HTTP Sessions

When a request contains a session information stored in a cookie, in the POST data, or by URL encoding, the session ID contains a reference to the specific server in which the session was originally established (called the *primary* server) and a reference to an additional server where the original session is replicated (called the *secondary* server). A request containing a cookie attempts to connect to the primary server If that attempt fails, the request is routed to the secondary server. If both the primary and secondary servers fail, the session is lost and the plug-in attempts to make a fresh connection to another server in the dynamic cluster list. For more information see Figure 10-1 “Connection Failover” on page 10-13.

Figure 10-1 Connection Failover



*The Maximum number of retries allowed in the red loop is equal to `ConnectTimeoutSecs` ÷ `ConnectRetrySecs`.

CHAPTER

11 Installing and Configuring the Netscape Enterprise Server Plug-In (NSAPI)

The following sections describe how to install and configure the Netscape Enterprise Server Plug-In (NES) proxy plug-in:

Overview of the Netscape Enterprise Server Plug-In

The Netscape Enterprise Server Plug-In enables requests to be proxied from Netscape Enterprise Server (NES, also called iPlanet) to WebLogic Server. The plug-in enhances an NES installation by allowing WebLogic Server to handle those requests that require the dynamic functionality of WebLogic Server.

The Netscape Enterprise Server Plug-In is designed for an environment where Netscape Enterprise Server serves static pages, and a Weblogic Server (operating in a different process, possibly on a different host or hosts) is delegated to serve dynamic pages, such as JSPs or pages generated by HTTP Servlets. The connection between WebLogic Server and the Netscape Enterprise Server Plug-In is made using clear text or Secure Sockets Layer (SSL). To the end user—the browser—the HTTP requests delegated to WebLogic Server appear to come from the same source as the static pages. Additionally, the HTTP-tunneling facility of the WebLogic Server can operate through the Netscape Enterprise Server Plug-In, providing access to all WebLogic Server services (not just dynamic pages).

The Netscape Enterprise Server Plug-In operates as an NSAPI module (see `http://home.netscape.com/servers/index.html`) within a Netscape Enterprise Server. The NSAPI module is loaded by NES at startup, and then certain HTTP requests are delegated to it. NSAPI is similar to an HTTP (Java) servlet, except that a NSAPI module is written in code native to the platform.

For more information on supported versions of Netscape Enterprise Server and iPlanet servers, see the BEA WebLogic Server Platform Support Page at `http://e-docs.bea.com/wls/platforms/index.html#plugin`.

Connection Pooling and Keep-Alive

The WebLogic Server NSAPI plug-in provides efficient performance by using a re-usable pool of connections from the plug-in to WebLogic Server. The NSAPI plug-in automatically implements "keep-alive" connections between the plug-in and WebLogic Server. If a connection is inactive for more than 30 seconds or a user-defined amount of time, the connection is closed.

Proxying Requests

The plug-in proxies requests to WebLogic Server based on a configuration that you specify. You can proxy requests either based on the URL of the request (or a portion of the URL). This is called proxying by *path*. You can also proxy request based on the *MIME type* of the requested file. You can also use a combination of both methods. If a request matches both criteria, the request is proxied by path. You can also specify additional parameters for each of these types of requests that define additional behavior of the plug-in. For more information, see the next section.

Installing and Configuring the Netscape Enterprise Server Plug-In

To install and configure the Netscape Enterprise Server Plug-In:

1. Copy the library.

   The WebLogic NSAPI plug-in module is distributed as a shared object (`.so`) on UNIX platforms and as a dynamic-link library (`.dll`) on Windows. These files are respectively located in the `/lib` or `/bin` directories of your WebLogic Server distribution.

Choose the appropriate library file for your environment from the BEA WebLogic Server Platform Support Page at `http://e-docs.bea.com/wls/platforms/index.html#plugin`.. and copy that file into the file system where NES is located.

2. Modify the `obj.conf` file. The `obj.conf` file defines which requests are proxied to WebLogic Server and other configuration information. For details see "Modifying the obj.conf File" on page 11-5.

3. If you are proxying requests by MIME type:

   a. Add the appropriate lines to the `obj.conf` file. For more information, see "Modifying the obj.conf File" on page 11-5.

   b. Add any new MIME types referenced in the `obj.conf` file to the `MIME.types` file. You can add MIME types by using the Netscape server console or by editing the `MIME.types` file directly.

      To directly edit the `MIME.types` file, open the file for edit and type the following line:

      ```
      type=text/jsp          exts=jsp
      ```

      Note: For NES 4.0 (iPlanet), instead of adding the MIME type for JSPs, change the existing MIME type from

      `magnus-internal/jsp`

      to

      `text/jsp.`

      To use the Netscape console, select Manage Preferences→ Mime Types, and make the additions or edits.

4. Deploy and test the Netscape Enterprise Server Plug-In

   a. Start WebLogic Server.

   b. Start Netscape Enterprise Server. If NES is already running, you must either restart it or apply the new settings from the console in order for the new settings to take effect.

   c. To test the Netscape Enterprise Server Plug-In, open a browser and set the URL to the Enterprise Server + `/weblogic/`, which should bring up the default WebLogic Server HTML page, welcome file, or default servlet, as defined for the default Web Application on WebLogic Server, as shown in this example:

```
http://myenterprise.server.com/weblogic/
```

Modifying the obj.conf File

To use the Netscape Enterprise Server Plug-In, you must make several modifications to the NES `obj.conf` file. These modifications specify how requests are proxied to WebLogic Server. You can proxy requests by URL or by MIME type. The procedure for each is described later in this section.

The Netscape `obj.conf` file is very strict about the placement of text. To avoid problems, note the following regarding the `obj.conf` file:

- Eliminate extraneous leading and trailing white space. Extra white space can cause your Netscape server to fail.
- If you must enter more characters than you can fit on one line, place a backslash (\) at the end of that line and continue typing on the following line. The backslash directly appends the end of the first line to the beginning of the following line. If a space is necessary between the words that end the first line and begin the second line, be certain to use *one* space, either at the end of the first line (before the backslash), or at the beginning of the second line.
- Do not split attributes across multiple lines. (For example, all servers in a cluster must be listed in the same line, following `WebLogicCluster`).
- If a required parameter is missing from the configuration, when the object is invoked it issues an HTML error that notes the missing parameter from the configuration.

To configure the `obj.conf` file:

1. Locate and open `obj.conf`.

   The `obj.conf` file for your NES instance is in the following location:

   `NETSCAPE_HOME``/https-`*`INSTANCE_NAME`*`/config/obj.conf`

   Where *`NETSCAPE_HOME`* is the root directory of the NES installation, and *`INSTANCE_NAME`* is the particular “instance” or server configuration that you are using. For example, on a UNIX machine called `myunixmachine`, the `obj.conf` file would be found here:

```
/usr/local/netscape/enterprise-351/
 https-myunixmachine/config/obj.conf
```

2. Instruct NES to load the native library as an NSAPI module

   Add the following lines to the beginning of the `obj.conf` file. These lines instruct NES to load the native library (the `.so` or `.dll` file) as an NSAPI module:

   ```
   Init fn="load-modules" funcs="wl_proxy,wl_init"\
   shlib=/usr/local/netscape/plugins/SHARED_LIBRARY
   Init fn="wl_init"
   ```

   Where *`SHARED_LIBRARY`* is the shared object or `dll` (for example `libproxy.so`) that you installed in step 1. under "Installing and Configuring the Netscape Enterprise Server Plug-In" on page 11-3. The function "`load-modules`" tags the shared library for loading when NES starts up. The values "`wl_proxy`" and "`wl_init`" identify the functions that the Netscape Enterprise Server Plug-In executes.

3. If you want to proxy requests by URL, (Also called proxying by *path*. If you want to proxy requests by MIME type, see step 4.) create a separate `<Object>` tag for each URL that you want to proxy and define the `PathTrim` parameter. Proxying by path supersedes proxying by MIME type. The following is an example of an `<Object>` tag that proxies a request containing the string `*/weblogic/*`.

   ```
   <Object name="weblogic" ppath="*/weblogic/*">
   Service fn=wl_proxy WebLogicHost=myserver.com\
    WebLogicPort=7001 PathTrim="/weblogic"
   </Object>
   ```

   To create an `<Object>` tag to proxy requests by URL:

   a. Specify a name for this object (optional) inside the opening `<Object>` tag using the `name` attribute. The `name` attribute is informational only and is not used by the Netscape Enterprise Server Plug-In. For example

      ```
      <Object name=myObject ...>
      ```

   b. Specify the URL to be proxied within the `<Object>` tag, using the `ppath` attribute. For example:

      ```
      <Object name=myObject ppath="*/weblogic/*>
      ```

      The value of the `ppath` attribute can be any string that identifies requests intended for Weblogic Server. When you use a `ppath`, every request that contains that path is redirected. For example, a `ppath` of "`*/weblogic/*`"

redirects every request that begins "`http://enterprise.com/weblogic`" to the Netscape Enterprise Server Plug-In, which sends the request to the specified Weblogic host or cluster.

c. Add the `Service` directive within the `<Object>` and `</Object>` tags. In the `Service` directive you can specify any valid parameters as name=value pairs. Separate multiple name=value pairs with *one and only one* space. For example:

```
Service fn=wl_proxy WebLogicHost=myserver.com\
 WebLogicPort=7001 PathTrim="/weblogic"
```

For a complete list of parameters, see "General Parameters for Web Server Plug-Ins" on page C-2. You *must* specify the following parameters:

For a *non-clustered* WebLogic Server:
The `WebLogicHost` and `WebLogicPort` parameters.

For a *cluster* of WebLogic Server:
The `WebLogicCluster` parameter.

The `Service` directive should always begin with Service `fn=wl_proxy`, followed by valid name=value pairs of parameters.

Here is an example of the object definitions for two separate `ppaths` that identify requests to be sent to different instances of WebLogic Server.

```
<Object name="weblogic" ppath="*/weblogic/*">
Service fn=wl_proxy WebLogicHost=myserver.com\
 WebLogicPort=7001 PathTrim="/weblogic"
</Object>
```

```
<Object name="si" ppath="*/servletimages/*">
Service fn=wl_proxy WebLogicHost=otherserver.com\
 WebLogicPort=7008
</Object>
```

Note: Parameters that are not required, such as `PathTrim`, can be used to further configure the way the `ppath` is passed through the Netscape Enterprise Server Plug-In. For a complete list of plug-in parameters, see "General Parameters for Web Server Plug-Ins" on page C-2.

4. If you want to proxy by MIME type, the MIME type must be listed in the `MIME.types` file. For instructions on modifying this file, see step 3. under "Installing and Configuring the Netscape Enterprise Server Plug-In" on page 11-3.

   All requests with a designated MIME type extension (for example, `.jsp`) can be proxied to the WebLogic Server, regardless of the URL.

To proxy all requests of a certain file type to WebLogic Server:

a. Add a `Service` directive to the existing `default Object` definition. (`<Object name=default ...>`).

   For example, to proxy all JSPs to a WebLogic Server, the following `Service` directive should be added *after* the last line that begins with:

   ```
   NameTrans fn=....
   ```

   and *before* the line that begins with:

   ```
   PathCheck.
   ```

   ```
   Service method="(GET|HEAD|POST|PUT)" type=text/jsp
   fn=wl_proxy\
    WebLogicHost=192.1.1.4 WebLogicPort=7001
   PathPrepend=/jspfiles
   ```

   This `Service` directive proxies all files with the `.jsp` extension to the designated WebLogic Server, where they are served with a URL like this:

   ```
   http://WebLogic:7001/jspfiles/myfile.jsp
   ```

   The value of the `PathPrepend` parameter should correspond to the context root of a Web Application that is deployed on the WebLogic Server or cluster to which requests are proxied.

   After adding entries for the Netscape Enterprise Server Plug-In, the default `Object` definition will be similar to the following example, with the additions shown in **`bold`**:

   ```
   <Object name=default>
   NameTrans fn=pfx2dir from=/ns-icons\
    dir="c:/Netscape/SuiteSpot/ns-icons"
   NameTrans fn=pfx2dir from=/mc-icons\
    dir="c:/Netscape/SuiteSpot/ns-icons"
   NameTrans fn="pfx2dir" from="/help" dir=\
    "c:/Netscape/SuiteSpot/manual/https/ug"
   NameTrans fn=document-root root="c:/Netscape/SuiteSpot/docs"
   Service method="(GET|HEAD|POST|PUT)" type=text/jsp\
    fn=wl_proxy WebLogicHost=localhost WebLogicPort=7001\
    PathPrepend=/jspfiles
   PathCheck fn=nt-uri-clean
   PathCheck fn="check-acl" acl="default"
   PathCheck fn=find-pathinfo
   PathCheck fn=find-index index-names="index.html,home.html"
   ObjectType fn=type-by-extension
   ObjectType fn=force-type type=text/plain
   Service method=(GET|HEAD) type=magnus-internal/imagemap\
   ```

```
 fn=imagemap
Service method=(GET|HEAD) \
 type=magnus-internal/directory fn=index-common
Service method=(GET|HEAD) \
 type=*~magnus-internal/* fn=send-file
AddLog fn=flex-log name="access"
</Object>
```

b. Add a similar `Service` statement to the default object definition for all other MIME types that you want to proxy to WebLogic Server.

5. If you want to enable HTTP-tunneling (optional):

   Add the following object definition to the `obj.conf` file, substituting the WebLogic Server host name and the WebLogic Server port number, or the name of a WebLogic Cluster that you wish to handle HTTP tunneling requests.

```
<Object name="tunnel" ppath="*/HTTPClnt*">
Service fn=wl_proxy WebLogicHost=192.192.1.4\
 WebLogicPort=7001
</Object>
```

Using SSL with the NSAPI Plug-In

You can use the Secure Sockets Layer (SSL) protocol to protect the connection between the Netscape Enterprise Server Plug-In, and WebLogic Server. The SSL protocol provides confidentiality and integrity to the data passed between the Netscape Enterprise Server Plug-In and WebLogic Server. In addition, the SSL protocol allows the WebLogic Server proxy plug-in to authenticate itself to the Netscape Enterprise Server to ensure that information is passed to a trusted principal.

The WebLogic Server proxy plug-in does *not* use the transport protocol (`http` or `https`) specified in the HTTP request (usually by the browser) to determine whether or not the SSL protocol will be used to protect the connection between the Netscape Enterprise Server Plug-In and WebLogic Server.

Note: You cannot configure a 2-way SSL between the Netscape Enterprise Server and WebLogic Server. The SSL protocol is a point-to-point connection, cyptographically sealed end-to-end. Therefore, any type of proxy or firewall cannot see into the SSL socket. The Netscape Enterprise Server acts as the server end-point in the SSL connection. The configuration is:

```
client-->2-way SSL-->NSAPI<--1-way SSL<--WebLogic Server
```

The Netscape Enterprise Server cannot use the digital certificate from the first SSL connection in the second SSL connection because it cannot use the client's private key.

To use the SSL protocol between Netscape Enterprise Server Plug-In and WebLogic Server:

1. Configure WebLogic Server for SSL. For more information, see "Configuring the SSL Protocol" on page 12-28.
2. Configure the WebLogic Server SSL listen port. For more information, see "Configuring the Listen Port" on page 7-3.
3. Set the `WebLogicPort` parameter in the `Service` directive in the `obj.conf` file to the listen port configured in step 2.
4. Set the `SecureProxy` parameter in the `Service` directive in the `obj.conf` file file to `ON`.
5. Set additional parameters in the `Service` directive in the `obj.conf` file that define information about the SSL connection. For a complete list of parameters, see "SSL Parameters for Web Server Plug-Ins" on page C-11.

Connection Errors and Clustering Failover

When the Netscape Enterprise Server Plug-In attempts to connect to WebLogic Server, the plug-in uses several configuration parameters to determine how long to wait for connections to the WebLogic Server host and, after a connection is established, how long the plug-in waits for a response. If the plug-in cannot connect or does not receive a response, the plug-in will attempt to connect and send the request to other WebLogic Servers in the cluster. If the connection fails or there is no response from any WebLogic Server in the cluster, an error message is sent.

Figure 11-1 “Connection Failover” on page 11-13 demonstrates how the plug-in handles failover.

Connection Failures

Failure of the host to respond to a connection request could indicate possible problems with the host machine, networking problems, or other server failures.

Failure of WebLogic Server to respond, could indicate that WebLogic Server is not running or is unavailable, a hung server, a database problem, or other application failure.

Failover with a Single, Non-Clustered WebLogic Server

If you are running only a single WebLogic Server the same logic described here applies, except that the plug-in only attempts to connect to the server defined with the `WebLogicHost` parameter. If the attempt fails, an `HTTP 503` error message is returned. The plug-in continues trying to connect to WebLogic Server until `ConnectTimeoutSecs` is exceeded.

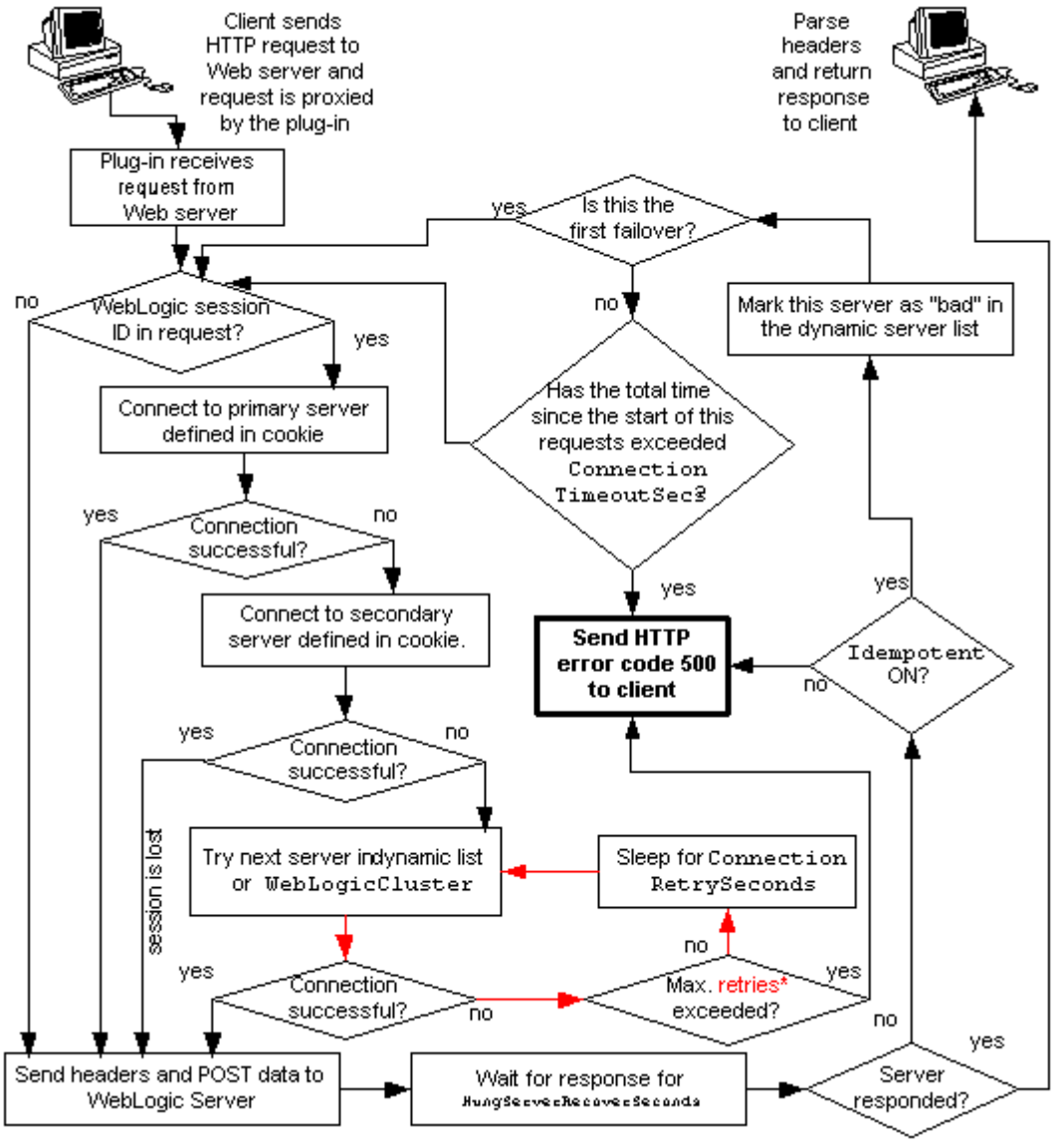
The Dynamic Server List

When you specify a list of WebLogic Servers in the `WebLogicCluster` parameter, the plug-in uses that list as a starting point for load balancing among the members of the cluster. After the first request is routed to one of these servers, a dynamic server list is returned containing an updated list of servers in the cluster. The updated list adds any new servers in the cluster and deletes any that are no longer part of the cluster or that have failed to respond to requests. This list is updated automatically with the HTTP response when a change in the cluster occurs.

Failover, Cookies, and HTTP Sessions

When a request contains a session information stored in a cookie, in the POST data, or by URL encoding, the session ID contains a reference to the specific server in which the session was originally established (called the *primary* server) and a reference to an additional server where the original session is replicated (called the *secondary* server). A request containing a cookie attempts to connect to the primary server If that attempt fails, the request is routed to the secondary server. If both the primary and secondary servers fail, the session is lost and the plug-in attempts to make a fresh connection to another server in the dynamic cluster list. For more information see Figure 11-1 “Connection Failover” on page 11-13.

Figure 11-1 Connection Failover



*The Maximum number of retries allowed in the red loop is equal to `ConnectTimeoutSecs ÷ ConnectRetrySecs.`

Failover Behavior When Using Firewalls and Load Directors

In most configurations, the Netscape Enterprise Server Plug-In sends a request to the primary instance of a cluster. When that instance is unavailable, the request fails over to the secondary instance. However, in some configurations that use combinations of firewalls and load-directors, any one of the servers (firewall or load-directors) can accept the request and return a successful connection while the primary instance of WebLogic Server is unavailable. After attempting to direct the request to the primary instance of WebLogic Server (which is unavailable), the request is returned to the plug-in as "connection reset".

Requests running through combinations of firewalls (with or without load-directors) are handled by WebLogic Server. In other words, responses of `connection reset` fail over to a secondary instance of WebLogic Server. Because responses of `connection reset` fail over in these configurations, servlets must be idempotent. Otherwise duplicate processing of transactions may result.

Sample obj.conf file (not using a WebLogic Cluster)

Below is an example of lines which should be added to the `obj.conf` file if you are not using a cluster. You can use this example as a template that you can modify to suit your environment and server. Lines beginning with'#' are comments.

Note: Make sure that you do not include any extraneous white space in the `obj.conf` file. Copying and pasting from the samples below sometimes adds extra white space, which can create problems when reading the file.

You can read the full documentation on Enterprise Server configuration files in the Netscape Enterprise Server Plug-In documentation.

```
## ------------- BEGIN SAMPLE OBJ.CONF CONFIGURATION  ---------
# (no cluster)

# The following line locates the NSAPI library for loading at
# startup, and identifies which functions within the library are
# NSAPI functions.  Verify the path to the library (the value
# of the shlib=<...> parameter) and that the file is
# readable, or the server fails to start.

Init fn="load-modules" funcs="wl_proxy,wl_init"\
 shlib=/usr/local/netscape/plugins/libproxy.so
Init fn="wl_init"

# Configure which types of HTTP requests should be handled by the
# NSAPI module (and, in turn, by WebLogic).  This is done
# with one or more "<Object>" tags as shown below.

# Here we configure the NSAPI module to pass requests for
# "/weblogic" to a WebLogic Server listening at port 7001 on
# the host myweblogic.server.com.

<Object name="weblogic" ppath="*/weblogic/*">
Service fn=wl_proxy WebLogicHost=myweblogic.server.com\
 WebLogicPort=7001 PathTrim="/weblogic"
</Object>

# Here we configure the plug-in so that requests that
# match "/servletimages/" is handled by the
# plug-in/WebLogic.
```

```
<Object name="si" ppath="*/servletimages/*">
Service fn=wl_proxy WebLogicHost=192.192.1.4 WebLogicPort=7001
</Object>

# This Object directive works by file extension rather than
# request path. To use this configuration, you must also add
# a line to the mime.types file:
#
# type=text/jsp             exts=jsp
#
# This configuration means that any file with the extension
# ".jsp" are proxied to WebLogic. Then you must add the
# Service line for this extension to the Object "default",
# which should already exist in your obj.conf file:

<Object name=default>
NameTrans fn=pfx2dir from=/ns-icons\
 dir="c:/Netscape/SuiteSpot/ns-icons"
NameTrans fn=pfx2dir from=/mc-icons\
 dir="c:/Netscape/SuiteSpot/ns-icons"
NameTrans fn="pfx2dir" from="/help" dir=\
 "c:/Netscape/SuiteSpot/manual/https/ug"
NameTrans fn=document-root root="c:/Netscape/SuiteSpot/docs"
Service method="(GET|HEAD|POST|PUT)" type=text/jsp fn=wl_proxy\
 WebLogicHost=localhost WebLogicPort=7001 PathPrepend=/jspfiles
PathCheck fn=nt-uri-clean
PathCheck fn="check-acl" acl="default"
PathCheck fn=find-pathinfo
PathCheck fn=find-index index-names="index.html,home.html"
ObjectType fn=type-by-extension
ObjectType fn=force-type type=text/plain
Service method=(GET|HEAD) type=magnus-internal/imagemap\
 fn=imagemap
Service method=(GET|HEAD) \
 type=magnus-internal/directory fn=index-common
Service method=(GET|HEAD) type=*~magnus-internal/* fn=send-file
AddLog fn=flex-log name="access"
</Object>

# The following directive enables HTTP-tunneling of the
# WebLogic protocol through the NSAPI plug-in.

<Object name="tunnel" ppath="*/HTTPClnt*">
Service fn=wl_proxy WebLogicHost=192.192.1.4 WebLogicPort=7001
</Object>

#
## ------------- END SAMPLE OBJ.CONF CONFIGURATION ---------
```

Sample obj.conf file (using a WebLogic Cluster)

Below is an example of lines which should be added to `obj.conf` if you are using a WebLogic Server cluster. You can use this example as a template that you can modify to suit your environment and server. Lines beginning with # are comments.

Note: Make sure that you do not include any extraneous white space in the `obj.conf` file. Copying and pasting from the samples below sometimes adds extra white space, which can create problems when reading the file.

For more information, see the full documentation on Enterprise Server configuration files from Netscape.

```
## ------------- BEGIN SAMPLE OBJ.CONF CONFIGURATION ---------
# (using a WebLogic Cluster)
#
# The following line locates the NSAPI library for loading at
# startup, and identifies which functions within the library are
# NSAPI functions.  Verify the path to the library (the value
# of the shlib=<...> parameter) and that the file is
# readable, or the server fails to start.

Init fn="load-modules" funcs="wl_proxy,wl_init"\
 shlib=/usr/local/netscape/plugins/libproxy.so
Init fn="wl_init"

# Configure which types of HTTP requests should be handled by the
# NSAPI module (and, in turn, by WebLogic).  This is done
# with one or more "<Object>" tags as shown below.

# Here we configure the NSAPI module to pass requests for
# "/weblogic" to a cluster of WebLogic Servers.

<Object name="weblogic" ppath="*/weblogic/*">
Service fn=wl_proxy \
 WebLogicCluster="myweblogic.com:7001,yourweblogic.com:7001,\
 theirweblogic.com:7001" PathTrim="/weblogic"
</Object>

# Here we configure the plug-in so that requests that
# match "/servletimages/" are handled by the
# plug-in/WebLogic.
```

```
<Object name="si" ppath="*/servletimages/*">
Service fn=wl_proxy \
WebLogicCluster="myweblogic.com:7001,yourweblogic.com:7001,\
 theirweblogic.com:7001"
</Object>

# This Object directive works by file extension rather than
# request path. To use this configuration, you must also add
# a line to the mime.types file:
#
# type=text/jsp             exts=jsp
#
# This configuration means that any file with the extension
# ".jsp" is proxied to WebLogic. Then you must add the
# Service line for this extension to the Object "default",
# which should already exist in your obj.conf file:

<Object name=default>
NameTrans fn=pfx2dir from=/ns-icons\
 dir="c:/Netscape/SuiteSpot/ns-icons"
NameTrans fn=pfx2dir from=/mc-icons\
 dir="c:/Netscape/SuiteSpot/ns-icons"
NameTrans fn="pfx2dir" from="/help" dir=\
 "c:/Netscape/SuiteSpot/manual/https/ug"
NameTrans fn=document-root root="c:/Netscape/SuiteSpot/docs"
Service method="(GET|HEAD|POST|PUT)" type=text/jsp fn=wl_proxy\
 WebLogicCluster="myweblogic.com:7001,yourweblogic.com:7001,\
 theirweblogic.com:7001",PathPrepend=/jspfiles
PathCheck fn=nt-uri-clean
PathCheck fn="check-acl" acl="default"
PathCheck fn=find-pathinfo
PathCheck fn=find-index index-names="index.html,home.html"
ObjectType fn=type-by-extension
ObjectType fn=force-type type=text/plain
Service method=(GET|HEAD) type=magnus-internal/imagemap\
 fn=imagemap
Service method=(GET|HEAD) \
 type=magnus-internal/directory fn=index-common
Service method=(GET|HEAD) type=*~magnus-internal/* fn=send-file
AddLog fn=flex-log name="access"
</Object>

# The following directive enables HTTP-tunneling of the
# WebLogic protocol through the NSAPI plug-in.

<Object name="tunnel" ppath="*/HTTPClnt*">
Service fn=wl_proxy WebLogicCluster="myweblogic.com:7001,\
 yourweblogic.com:7001,theirweblogic.com:7001"
</Object>
```

```
#
## ------------- END SAMPLE OBJ.CONF CONFIGURATION ---------
```

CHAPTER

12 Managing Security

This section discusses the following topics:

- Overview of Configuring Security
- Changing the System Password
- Specifying a Security Realm
- Defining Users
- Defining Groups
- Defining a Group for a Virtual Host
- Defining ACLs
- Configuring the SSL Protocol
- Configuring Mutual Authentication
- Configuring RMI over IIOP over SSL
- Protecting Passwords
- Installing an Audit Provider
- Installing a Connection Filter
- Configuring Security Context Propagation
- Setting Up the Java Security Manager
- Modifying the weblogic.policy File for Third Party or User-Written Classes

Overview of Configuring Security

Implementing security in a WebLogic Server deployment largely consists of configuring fields that define the security policy for that deployment. WebLogic Server provides an Administration Console to help you define the security policy for your deployment. Using the Administration Console, specify security-specific values for the following elements of your deployment:

- Realms
- Users and Groups
- Access Control Lists (ACLs) and permissions for WebLogic Server resources
- SSL protocol
- Mutual authentication
- Audit providers
- Custom filters
- Security context propagation

Because security features are closely related, it is difficult to determine where to start when configuring security. In fact, defining security for your WebLogic Server deployment may be an iterative process. Although more than one sequence of steps may work, we recommend the following procedure:

1. Change the system password to protect your WebLogic Server deployment.
2. Specify a security realm. By default, WebLogic Server is installed with the File realm in place. However, you may prefer an alternate security realm or a custom security realm.
3. Define Users for the security realm. You can organize Users further by implementing Groups in the security realm.
4. Define ACLs and permissions for the resources in your WebLogic Server deployment.

5. Protect the network connection between clients and WebLogic Server by implementing the SSL protocol. When SSL is implemented, WebLogic Server uses its digital certificate, issued by a trusted certificate authority, to authenticate clients. This step is an optional but we recommend it.
6. Further protect your WebLogic Server deployment by implementing mutual authentication. When mutual authentication is implemented, WebLogic Server must authenticate itself to the client and then the client in turn, must authenticate itself to WebLogic Server. Again, this step is an optional but we recommend it.

This section describes these configuration steps and the fields you set in the Administration Console. For a complete description of WebLogic Server security features, see Introduction to WebLogic Security and Security Fundamentals.

For information about setting the security fields in the Administration Console and detailed descriptions of each field, see the Administration Console Online Help.

For information about assigning security roles to WebLogic EJBs, see WebLogic Server 6.0 Deployment Properties.

For information about security in WebLogic web applications, see Deploying and Configuring Web Applications.

Changing the System Password

During installation you specify a password for the system User. The specified password is associated with the `system` User in WebLogic Server and is stored in the `fileRealm.properties` file in the `\wlserver6.0\config\`*`mydomain`* directory. The specified password corresponds to the Administration server for the domain and all the managed servers associated with that Administration server.

The password is encrypted and is further protected when WebLogic Server applies a hash to it. To improve security, we recommend frequently changing the system password that was set during installation.

To change the system password, do the following:

1. Open the Users window in the Administration Console.
2. Enter system in the User field.

3. Enter a new password in the Password field.
4. Confirm the password.

When using an Administration Server and managed servers in a domain, the managed server must always use the password for the Administration Server in the domain. Always change the password for the Administration Server through the Administration Console. The new password is propagated to all the managed server in the domain. Remember the system password for a domain must match across all the servers in the domain.

Note: The Petstore and the ExampleServer domains still store the system password in a `password.ini` file. When using these domains, modify the system password by modifying the password information in the `password.ini` file.

Specifying a Security Realm

By default WebLogic Server is installed with the File realm in place. Before using the File realm, you need to define several fields that govern the use of the File realm. You set these fields on the Filerealm tab in the Security window of the Administration Console.

The following table describes each field on the Filerealm tab.

Table 12-1 File Realm Fields

Field	Description
Max Users	Specifies the maximum number of Users to be used with the File realm. The File realm is intended to be used with 10,000 or fewer Users. The minimum value for this field is 1 and the maximum value is 10,000. The default is 1,000.
Max Groups	Specifies the maximum number of Groups to be used with the File realm**.** The minimum value for this field is 1 and the maximum value is 10,000. The default is 1,000.

Field	Description
Max ACLs	Specifies the maximum number of ACLs to be used with the File realm. The minimum value for this field is 1 and the maximum value is 10,000. The default is 1,000.

If for any reason, `fileRealm.properties` is corrupted or destroyed, you must reconfigure all the security information for WebLogic Server. Therefore, we recommend that you take the following steps:

- Make a backup copy of the `fileRealm.properties` file and put it in a secure place.
- Set the permissions on the `fileRealm.properties` file protections such that the administrator of the WebLogic Server deployment has write and read privileges and no other users have any privileges.

Note: You should also make a backup copy of the `SerializedSystemIni.dat` file for the File realm. For more information about the `SerializedSystemIni.dat` file, see Protecting Passwords

If, instead of the File realm, you want to use one of the alternate security realms provided by WebLogic Server or a custom security realm, set the fields for the desired realm and reboot WebLogic Server.

For more information about security realms in WebLogic Server, see Security Realms.

Configuring the Caching Realm

Note: All configuration instructions are based on the use of the Administration Console.

The Caching realm works with the File realm, alternate security realms, or custom security realms to fulfill client requests with the proper authentication and authorization. The Caching realm stores the results of both successful and unsuccessful realm lookups. It manages separate caches for Users, Groups, permissions, ACLs, and authentication requests. The Caching realm improves the performance of WebLogic Server by caching lookups, reducing the number of calls into other security realms. For more information about security realms in WebLogic Server, see Security Realms.

The Caching realm is installed automatically when you install WebLogic Server: the cache is set up to delegate to the other security realms but caching is not enabled. You have to enable caching through the Administration Console.

When you enable caching, the Caching realm saves the results of a realm lookup in its cache. Lookup results remain in the cache until either the specified number of seconds defined for the time-to-live (TTL) fields has passed (the lookup result has expired) or the cache has filled. When the cache is full, new lookup results replace the oldest cached results. The TTL fields determine how long a cached object is valid. The higher you set these fields, the less often the Caching realm calls the secondary security realm. Reducing the frequency of such calls improves the performance. The trade-off is that changes to the underlying security realm are not recognized until the cached object expires.

Note: When you obtain an object from a security realm, the object reflects a snapshot of the object. To update the object, you must call the object's `get()` method again. For example, the membership of a Group is set when the Group is retrieved from the security realm with a call to the `getgroup()` method. To update the members of the Group, you must call the `getgroup()` method again.

By default, the Caching realm operates on the assumption that the secondary security realm is case-sensitive. In a case-sensitive security realm, the owners of usernames `bill` and `Bill`, for example are treated as two distinct Users. The Windows NT Security realm and the LDAP Security realm are examples of security realms that are not case-sensitive. If you are using a security realm that is not case-sensitive, you must disable the CacheCaseSensitive field. When this field is set, the Caching realm converts usernames to lowercase so that WebLogic Server gives correct results for the security realm when it performs case-sensitive comparisons. When defining or referencing Users or Groups in a case-sensitive security realm, type usernames in lowercase.

Configuring the Caching realm involves enabling various types of caches (such as ACL, Authentication, Group, and Permission) and defining how each cache operates. To do these tasks, define values for the field shown on the General tab in the Caching Realm Configuration window. To save your changes, click the Apply button. When you have finished defining the fields, reboot WebLogic Server.

The following table describes each field on the General tab.

Table 12-2 Caching Realm Fields

Field	Description
Name	Displays the active security realm. This field can not be changed.
Basic Realm	The name of the class for the alternate security realm or custom security realm being used with the Caching realm.
Case Sensitive Cache	Defines whether the specified security realm is case-sensitive. By default, this field is enabled: the realm is case-sensitive. To use a realm that is not case-sensitive (such as the Windows NT and LDAP security realms), you must disable this field.

To enable and configure the ACL cache, define values for the fields shown on the ACL tab in the Caching Realm Configuration window. To save your changes, click the Apply button. When you have finished defining the fields, reboot WebLogic Server.

The following table describes each field on the ACL tab.

Table 12-3 ACL Cache Fields

Field	Description
Enable ACL Cache	Option for enabling the ACL cache.
ACL Cache Size	The maximum number of ACL lookups to cache. The default is 211. This field should be a prime number for best lookup performance.
ACL Cache Positive TTL	The number of seconds to retain the results of a successful lookup. The default is 60 seconds.
ACL Cache Negative TTL	The number of seconds to retain the results of an unsuccessful lookup. The default is 10 seconds.

To enable and configure the Authentication cache, define values for the fields shown on the Authentication tab in the Caching Realm Configuration window. To save your changes, click the Apply button. When you have finished defining the fields, reboot WebLogic Server.

The following table describes each field on the Authentication tab.

Table 12-4 Authentication Cache Fields

Field	Description
Enable Authentication Cache	Option for enabling the Authentication cache.
Authentication Cache Size	The maximum number of Authenticate requests to cache. The default is 211. This field should be a prime number for best lookup performance.
Authentication Cache TTLPositive	The number of seconds to retain the results of a successful lookup. The default is 60 seconds.
Authentication Cache TTLNegative	The number of seconds to retain the results of an unsuccessful lookup. The default is 10 seconds.

To enable and configure the Group cache, define values for the fields shown on the Groups tab in the Caching Realm Configuration window. To save your changes, click the Apply button. When you have finished defining the fields, reboot WebLogic Server.

The following table describes each field on the Group tab.

Table 12-5 Group Cache Fields

Field	Description
Group Cache Enable	Option for enabling the Group cache.
Group Cache Size	The maximum number of Group lookups to cache. The default is 211. This field should be a prime number for best lookup performance.

Table 12-5 Group Cache Fields

Field	Description
Group Cache TTLPositive	The number of seconds to retain the results of a successful lookup. The default is 60 seconds.
Group Cache TTLNegative	The number of seconds to retain the results of an unsuccessful lookup. The default is 10 seconds.
Group Membership Cache TTL	The number of seconds to store the members of a group before updating it. The default is 10 seconds.

To enable and configure the User cache, define values for the fields shown on the User tab in the Caching Realm Configuration window. To save your changes, click the Apply button. When you have finished defining the fields, reboot WebLogic Server.

The following table describes each field on the User tab.

Table 12-6 User Cache Fields

Field	Description
Enable User Cache	Option for enabling the User cache.
User Cache Size	The maximum number of User lookups to cache. The default is 211.This field should be a prime number for best lookup performance.
User Cache TTLPositive	The number of seconds to retain the results of a successful lookup. The default is 60 seconds.
User Cache TTLNegative	The number of seconds to retain the results of an unsuccessful lookup. The default is 10 seconds.

To enable and configure the Permission cache, define values for the field shown on the Permission tab in the Caching Realm Configuration window. To save your changes, click the Apply button. When you have finished defining the fields, reboot WebLogic Server.

The following table describes each field on the Permission tab.

Table 12-7 Permission Cache Fields

Field	Description
Enable Permission Cache	Option for enabling the Permission cache.
Permission Cache Size	The maximum number of Permission lookups to cache. The default is 211.This field should be a prime number for best lookup performance.
Permission Cache TTLPositive	The number of seconds to retain the results of a successful lookup. The default is 60 seconds.
Permission Cache TTLNegative	The number of seconds to retain the results of an unsuccessful lookup. The default is 10 seconds.

Configuring the LDAP Security Realm

Note: The LDAP security realm has been rewritten to provide improved performance and configurability. BEA recommends upgrading your WebLogic Server 6.0 installation to Service Pack 1.0 to take advantage of this functionality. WebLogic Server 6.0 Service Pack 1.0 is available from the BEA Systems Download page on the Web.

The LDAP Security realm provides authentication through a Lightweight Directory Access Protocol (LDAP) server. This server allows you to manage all the users for your organization in one place: the LDAP directory. The LDAP security realm has been tested against the following LDAP servers:

- Open LDAP

- Netscape Directory Server
- Microsoft Site Server

Although BEA cannot committ to supporting LDAP servers that have not been tested, the implementation of the LDAP security realm in WebLogic Server should work with most LDAP servers.

Configuring the LDAP Security realm involves defining fields that enable the LDAP Security realm in WebLogic Server to communicate with the LDAP server and fields that describe how Users and Groups are stored in the LDAP directory.

Before you can use the LDAP Security realm, you need to enable the Caching Realm and enter the class name of the LDAP Security realm in the Basic Realm field.

To use the LDAP Security realm instead of the File realm, go to the Security→Realms node in the left pane of the Administration Console. In the right pane of the Administration Console, click the Create a New LDAP Realm link.

To specify the name of the LDAP Security realm and the name of the class that contains the LDAP Security realm define values for the fields shown on the General tab in the LDAP Realm Create window. To save your changes, click the Apply button. When you have finished defining the fields, reboot WebLogic Server.

The following table describes each field n the General tab.

Table 12-8 LDAP Security Realm Fields on the General Tab

Field	Description
Name	The name of the LDAP Security realm such as AccountingRealm
Realm Class Name	The name of the Java class that contains the LDAP Security realm. The Java class should be included in the CLASSPATH of WebLogic Server.

To enable communication between the LDAP server and WebLogic Server define values for the fields shown on the LDAP tab in the LDAP Realm Create window. To save your changes, click the Apply button. When you have finished defining the fields, reboot WebLogic Server.

The following table describes each field on the General tab.

Table 12-9 LDAP Security Realm Fields on the LDAP Tab

Field	Description
LDAPURL	The location of the LDAP server. Change the URL to the name of the computer on which the LDAP server is running and the number of the port at which it is listening. If you want WebLogic Server to connect to the LDAP server using the SSL protocol, use the LDAP server's SSL port in the URL.
Principal	The distinguished name (DN) of the LDAP User used by WebLogic Server to connect to the LDAP server. This user must be able to list LDAP Users and Groups.
Credential	The password that authenticates the LDAP User, as defined in the Principal field.
Enable SSL	Option for enabling the use of the SSL protocol to protect communications between the LDAP server and WebLogic Server. Keep in mind the following guidelines: ■ Disable this field if the LDAP server is not configured to use the SSL protocol. ■ If you set the UserAuthentication field to `external`, this field must be enabled.
AuthProtocol	The type of authentication used to authenticate the LDAP server. Set this field to one of the following values: ■ `None` for no authentication ■ `Simple` for password authentication ■ `CRAM-MD5` for certificate authentication Netscape Directory Server supports `CRAM-MD5`. Microsoft Site Server and Novell NDS support `Simple`.

To specify how Users are stored in the LDAP directory define the fields shown on the Users tab in the LDAP Realm Create window. To save your changes, click the Apply button. When you have finished defining the fields, reboot WebLogic Server.

The following table describes each field on the Users tab.

Table 12-10 LDAP Security Realm Fields on the Users Tab

Field	Description
User Authentication	Determines the method for authenticating Users. Set this field to one of the following values: ■ `Bind` specifies that the LDAP security realm retrieves user data, including the password for the LDAP server, and checks the password in WebLogic Server. ■ `External` specifies that the LDAP Security realm authenticates a User by attempting to bind to the LDAP server with the username and password supplied by the WebLogic Server client. If you choose the `External` setting, you must also use the SSL protocol. ■ `Local` specifies that the LDAP security realm authenticates a user by looking up the UserPassword property in the LDAP directory and checking it against the passwords in WebLogic Server. When using Netscape Directory Server, this field needs to be set to `Bind`.
User Password Attribute	The password of the LDAP User.
User DN	A list of attributes that, when combined with the attributes in the `User Name Attribute` field, uniquely identifies an LDAP User.
User Name Attribute	The login name of the LDAP User. The value of this field can be the common name of an LDAP User but usually it is an abbreviated string, such as the User ID.

To specify how Groups are stored in the LDAP directory, assign values to the fields shown on the Groups tab in the LDAP Realm Create window. To save your changes, click the Apply button. When you have finished defining the fields, reboot WebLogic Server.

The following table describes each field on the Groups tab.

Table 12-11 LDAP Security Realm Field on the Groups Tab

Field	Description
Group DN	The list of attributes that, combined with the Group Name Attribute field, uniquely identifies a Group in the LDAP directory.
Group Name Attribute	The name of a Group in the LDAP directory. It is usually a common name.
Group Is Context	This Boolean checkbox specifies how Group membership is recorded in the LDAP directory. ■ Check this checkbox if each Group entry contains one User. By default, the field is enabled. ■ Uncheck this checkbox if there is one Group entry containing an attribute for each Group member.
Group Username Attribute	The name of the LDAP attribute that contains a Group member in a Group entry.

If you have enabled caching, the Caching realm caches Users and Groups internally to avoid frequent lookups in the LDAP directory. Each object in the Users and Groups caches has a TTL field that you set when you configure the Caching realm. If you make changes in the LDAP directory, those changes are not reflected in the LDAP Security realm until the cached object expires or is flushed from the cache. The default TTL is 60 seconds for unsuccessful lookups and 10 seconds for successful lookups. Unless you change the TTL fields for the User and Group caches, changes in the LDAP directory should be reflected in the LDAP Security realm in 60 seconds.

If some server-side code has performed a lookup in the LDAP Security realm, such as a `getUser()` call on the LDAP Security realm, the object returned by the realm cannot be released until the code releases it. Therefore, a User authenticated by WebLogic Server remains valid as long as the connection persists, even if you delete the user from the LDAP directory.

Configuring the Windows NT Security Realm

The Windows NT Security realm uses account information defined for a Windows NT domain to authenticate Users and Groups. You can view Users and Groups in the Windows NT Security realm through the Administration Console, but you must manage Users and Groups through the facilities provided by Windows NT.

The Windows NT Security realm provides authentication (Users and Groups) but not authorization (ACLs). The `system` User defined in WebLogic Server must also be declared in the Windows NT domain. On a Windows NT platform, WebLogic Server must be run under the `system` User account, and clients must supply the `system` User password to authenticate successfully. When you define the `system` User account in Windows NT, make sure the owner of the account has administrative privileges and can read security-related information from the Windows NT Domain controller.

To use the Windows NT Security realm, you must run WebLogic Server as a Windows NT Service on a computer in the Windows NT domain. You do not have to run WebLogic Server on a domain controller.

Because WebLogic Server reads ACLs from the `fileRealm.properties` file at startup time, you must restart WebLogic Server after you change an ACL. If you use Groups with your ACLs, you can reduce the frequency with which you must restart WebLogic Server. Changing the members of a Windows NT Group allows you to manage individual Users' access to WebLogic Server resources dynamically.

Before you can use the Windows NT Security realm, you need to enable the Caching Realm and enter the class name of the Windows NT Security realm in the Basic Realm field.

To use the Windows NT Security realm instead of the File realm, go to the Security→Realms node in the left pane of the Administration Console. In the right pane of the Administration Console, click the Create a New NT Realm link.

Configuring the Windows NT Security realm involves setting fields that define a name for the realm and the computer on which the Windows NT domain is running. To specify a realm name and computer, you must define values for the fields shown the NT Realm Create window of the Administration Console. To save your changes, click the Apply button. When you have finished defining the fields, reboot WebLogic Server.

The following table describes each field in the NT Realm Configuration window.

Table 12-12 Windows NT Security Realm Fields

Field	Description
Name	The name of the Windows NT Security realm, such as, AccountingRealm
Realm Class Name	The name of the Java class that implements the Windows NT Security realm. The Java class needs to be in the CLASSPATH of WebLogic Server.
Primary Domain	The host and port number of the computer where Users and Groups are defined for the Windows NT domain. If you enter multiple host and port numbers, use a comma delineated list.

Once you have configured the Windows NT Security realm in the Administration Console, you need to define the `system` User in Windows NT:

1. Use the Administrator account to log on to the Windows NT domain you are using with WebLogic Server.
2. Go to Programs→Administrative Tools.
3. Select User Manager.
4. Define the `system` User.
5. Check the `Show Advanced User Rights` option.
6. Select the `Act as part of the operating system` option from the Rights pull-down menu.
7. Check the Add button.
8. Make sure the Windows NT `PATH` environment variable includes the `\wlserver6.0\bin` directory. (WebLogic Server loads the `W1ntrealm.dll` from this directory.)

Configuring the UNIX Security Realm

The UNIX Security realm executes a small native program, `wlauth`, to look up Users and Groups and to authenticate Users on the basis of their UNIX login names and passwords. On some platforms, `wlauth` uses PAM (Pluggable Authentication Modules) which allows you to configure authentication services in the operating system without altering applications that use the service. On platforms for which PAM is not available, `wlauth` uses the standard login mechanism, including shadow passwords, where supported.

In UNIX, a user is defined as a member of a group in the following ways:

- The user is defined in a default group in `etc/passwd`.
- The user ID for a user is included in the `etc/group` entry for a specific group. The UNIX Security realm supports only this method of determining the members of a group.

Because WebLogic Server reads ACLs from the `fileRealm.properties` file at startup time, you must restart WebLogic Server after you change an ACL. If you use Groups with your ACLs, you can reduce the frequency with which you must restart WebLogic Server. Changing the members of a UNIX Group allows you to manage individual Users' access to WebLogic Server resources dynamically.

The `wlauth` program runs `setuid root`. You need root permissions to modify the ownership and file attributes on the `wlauth` program and to set up the PAM configuration file for `wlauth`.

Perform the following steps to configure the UNIX Security realm:

1. If WebLogic Server is installed on a network drive, copy the `wlauth` file to a file system on the computer that executes WebLogic Server, for example, the `/usr/sbin` directory. The `wlauth` file is in the `weblogic/lib/`*`arch`* directory, where *`arch`* is the name of your platform.
2. As the root user, run the following commands to change the `wlauth` owner and permissions:

   ```
   # chown root wlauth
   # chmod +xs wlauth
   ```

3. On PAM platforms (Solaris and Linux), set up the PAM configuration for `wlauth`.

Solaris—Add the following lines to your `/etc/pam.conf` file:

```
# Setup for WebLogic authentication on Solaris machines
#
wlauth auth required      /usr/lib/security/pam_unix.so.1
wlauth password required  /usr/lib/security/pam_unix.so.1
wlauth account required   /usr/lib/security/pam_unix.so.1
```

Linux—Create a file called `/etc/pam.d/wlauth` containing the following:

```
#%PAM-1.0
#
# File name:
# /etc/pam.d/wlauth
#
# If you do not use shadow passwords, delete "shadow".
auth required     /lib/security/pam_pwdb.so shadow
account required  /lib/security/pam_pwdb.so
```

Note: Omit `shadow` if you are not using shadow passwords.

To use the UNIX Security realm instead of the File realm, go to the Security→Realms node in the left pane of the Administration Console. In the right pane of the Administration Console, click the Create a New UNIX Realm link.

Before you can use the UNIX Security realm, you need to enable the Caching Realm and enter the class name of the UNIX Security realm in the Basic Realm field.

Configuring the UNIX Security realm involves setting fields that define a name for the realm and the program that provides authentication services for the UNIX Security realm.To define these names, specify values for the fields on the UNIX Realm Create window of the Administration Console. To save your changes, click the Apply button. When you have finished defining the fields, reboot WebLogic Server.

The following table describes each field in the UNIX Realm Create window.

Table 12-13 UNIX Security Realm Fields

Field	Description
Name	The name of the UNIX Security realm, such as, AccountingRealm
Realm Class Name	The name of the WebLogic class that implements the UNIX Security realm. The Java class needs to be in the CLASSPATH of WebLogic Server.

Table 12-13 UNIX Security Realm Fields

Field	Description
AuthProgram	The name of the program used to authenticate users in the UNIX security realm. In most cases, the name of the program is `wlauth`.

If `wlauth` is not in the WebLogic Server class path or if you have given the program a name other than `wlauth`, you must add a Java command-line property when you start WebLogic Server. Edit the script you use to start WebLogic Server and add the following option after the `java` command:

```
-Dweblogic.security.unixrealm.authProgram=wlauth_prog
```

Replace *`wlauth_prog`* with the name of the `wlauth` program, including the full path if the program is not in the search path. Start WebLogic Server. If the `wlauth` program is in the WebLogic Server path and is named `wlauth`, this step is not needed.

Configuring the RDBMS Security Realm

The RDBMS Security realm is a BEA-provided custom security realm that stores Users, Groups and ACLs in a relational database. The RDBMS Security realm can be managed through the Administration Console.

To use the RDBMS Security realm instead of the File realm, go to the Security→Realms node in the left pane of the Administration Console. In the right pane of the Administration Console, click the Create a New RDBMS Realm link.

Before you can use the RDBMS Security realm, you need to enable the Caching Realm and enter the class name of the RDBMS Security realm in the Basic Realm field.

Configuring the RDBMS Security realm involves setting fields that define the JDBC driver being used to connect to the database and defining the schema used to store Users, Groups, and ACLs in the database.

To define these fields, you must specify values for the fields shown on three tabs of the RDBMS Realm Create window: the General tab, the Database tab, and the Schema tab. The following table describes each field that you must set on the General tab.

Table 12-14 RDBMS Security Realm Fields on the General Tab

Field	Description
Name	The name of the RDBMS Security realm, such as, AccountingRealm
Realm Class	The name of the WebLogic class that implements the RDBMS Security realm. The Java class needs to be in the CLASSPATH of WebLogic Server.

The following table describes the fields you must set on the Database tab.

Table 12-15 RDBMS Security Realm Fields on the Database Tab

Field	Description
Driver	The full class name of the JDBC driver. This class name must be in the CLASSPATH of WebLogic Server.
URL	The URL for the database you are using with the RDBMS realm, as specified by your JDBC driver documentation.
User Name	The default user name for the database.
Password	The password for the default user of the database.

The Schema properties used to define the Users, Groups, and ACLs stored in the database are listed on the Schema tab. When you finish defining values for the necessary fields on each of the three tabs, save your changes by clicking the Apply button. Then reboot WebLogic Server.

Installing a Custom Security Realm

You can create a custom security realm that draws from an existing store of Users such as directory server on the network. To use a custom security realm, you create an implementation of the `weblogic.security.acl.AbstractListableRealm` interface or the `weblogic.security.acl.AbstractManageableRealm` interface and then use the Administration Console to install your implementation.

To install a custom security realm, go to the Security→Realms node in the left pane of the Administration Console. In the right pane of the Administration Console, click the Create a New Custom Realm link.

Before you can use a custom security realm, you need to enable the Caching Realm and enter the class name of the custom security realm in the Basic Realm field.

Configuring a custom security realm involves setting fields that define a name for the realm and the interface that implements the realm, and specifying information that defines how the Users, Groups, and optionally ACLs are stored in the custom security realm. To define this information, you must specify values for the fields of the Custom Realm Create window of the Administration Console. To save your changes, click the Apply button. When you have finished defining the fields, reboot WebLogic Server.

The following table describes the fields you must set on the Custom Security Realm Create window.

Table 12-16 Custom Security Realm Fields

Field	Description
Name	The name of the Custom Security realm, such as, AccountingRealm
Realm Class Name	The name of the WebLogic class that implements the Custom Security realm. The Java class needs to be in the CLASSPATH of WebLogic Server.
Configuration Data	The information needed to connect to the security store.

For information about writing a custom security realm, see Writing a Custom Security Realm.

Testing an Alternate Security Realm or a Custom Security Realm

If you have started WebLogic Server with an alternate or a custom security realm, perform the following steps to ensure the realm is working properly:

1. Start the Administration Console. The Administration Console displays all the Users, Groups, and ACLs known in the security realm.
2. Use the Administration Console to add an ACL for the HelloWorld example servlet. Give a User and a Group in your security realm access to the HelloWorld example servlet. Select a Group that does not include the specified User.
3. Restart WebLogic Server and load the HelloWorld servlet with an ACL using the following URL:

   `http://localhost:portnumber/helloWorld`

   Try entering the username and password for a User who is not included in the ACL you added for the servlet. You should get a message telling you that you are not authorized to do so.

   Try entering the username and password of a User who is included in the ACL, either as an individual User or as a member of the Group. The servlet should load and display the Hello World message.

Migrating Security Realms

WebLogic Server 6.0 provides a new management architecture for security realms. The management architecture implemented through MBeans allows you to manage security realms through the Administration Console. If you have a security realm from a previous release of WebLogic Server, use the following information to migrate to the new architecture:

- If you are using the Windows NT, UNIX, or LDAP security realms, use the Convert WebLogic Properties option in the Administration Console to convert the security realm to the new architecture. Note that you can view Users, Groups, and ACLs in a Windows NT, UNIX, or LDAP security realm in the Administration Console, however, you still need to use the tools in the Windows NT, UNIX, or LDAP environments to manage Users and Groups.

- If you are using a custom security realm, follow the steps in "Installing a Custom Security Realm" to specify information about how the Users, Groups, and optionally ACLs are stored in your custom security realm.
- The Delegating security realm is no longer support in WebLogic Server 6.0. If you are using the Delegating security realm, you will have to use another type of security realm to store Users, Groups, and ACLs.
- If you are using the RDBMS security realm, use one of the following options to convert the security realm:
  - If you did not change the source for the RDBMS security realm, follow the steps in "Configuring the RDBMS Security Realm" to instantiate a new class for your existing RDBMS security realm and define information about the JDBC driver being used to connect to the database and the schema used by the security realm. In this case, you are creating a MBean in WebLogic Server 6.0 for the RDBMS security realm.
  - If you customized the RDBMS security realm, convert your source to use the MBeans. Use the code example in the `\samples\examples\security\rdbmsrealm` directory as a guide to converting your RDBMS security realm. Once you have converted your RDBMS security realm to MBeans, follow the instructions in"Configuring the RDBMS Security Realm" to define information about the JDBC driver being used to connect to the database and the schema used by the security realm.

Defining Users

Note: This section explains how to add Users to the File realm. If you are using an alternate security realm, you must use the management tools provided in that realm to define a User.

Users are entities that can be authenticated in a WebLogic Server security realm. A User can be a person or a software entity, such as a Java client. Each User is given a unique identity within a WebLogic Server security realm. As a system administrator you must guarantee that no two Users in the same security realm are identical.

Defining Users in a security realm involves specifying a unique name and password for each User that will access resources in the WebLogic Server security realm in the Users window of the Administration Console.

The following table describes the fields in the Users window.

Table 12-17 User Fields

Field	Description
Name	The name of a User, that is, an entity that will access WebLogic Server resources. Names are case-sensitive.
Password	The password for the User. The password must contain a minimum of 8 characters in length. Passwords are case-sensitive.

The File realm has two special users, `system` and `guest`:

- The `system` User is the administrative user who controls system-level WebLogic Server operations, such as starting and stopping servers, and locking and unlocking resources. The `system` User is defined during the WebLogic Server installation procedure.
- The `guest` User is automatically provided by WebLogic Server. When authorization is not required, WebLogic Server assigns the `guest` identity to a client this giving the client access to any resources that are available to the `guest` user. A client can log in as the `guest` User by entering `guest` as the username and `guest` as the password when prompted by a Web browser or by supplying the `guest` username and password in a Java client.

The `system` and `guest` Users are like other Users in a WebLogic Server security realm:

- To access WebLogic Server resources, they must have appropriate ACLs.
- To execute an operation on a WebLogic Server resource, they must provide a username and password (or digital certificate).

To improve the security of your WebLogic Server deployment, we recommend disabling the `guest` User. To do so, check the `Guest Disabled option` on the General tab in the Security window in the Administration Console. When you disable the `guest` User, the `guest` User is not deleted rather it just becomes unavailable so that no one can log on as the `guest` User.

To delete Users, enter the name of the User in the Remove These Users list box and click Remove.

For more information about Users and the access control model in WebLogic Server, see Introduction to WebLogic Security and Security Fundamentals.

Defining Groups

Note: This section explains how to add Groups to the File realm. If you are using an alternate security realm, you need to use the management tools provided in that realm to define a Group.

A Group represents a set of Users who usually have something in common, such as working in the same department in a company. Groups are used primarily to manage a number of Users in an efficient manner. When a Group is granted a permission in an ACL, all members of the Group effectively receive that permission.

You can register a Group with the WebLogic Server security realm by performing the following steps:

1. Setting the Name field in the Groups window of the Administration Console.
2. Clicking the Create button.
3. Entering Users in the Add User field.
4. Clicking the Update Group button when you finish adding Users.

The File realm has one built-in Group: `everyone`. All Users defined in the defined security realm are automatically members of the `everyone` Group.

To delete Groups, enter the name of the Group in the Remove These Groups list box and click Remove.

For more information about Groups and the access control model in WebLogic Server, see Introduction to WebLogic Security and Security Fundamentals.

Defining a Group for a Virtual Host

In WebLogic Server, virtual hosts that require authentication are represented in a security realm as a group. All the users of the virtual host are defined first as users of the security realm for a particular WebLogic Server and then defined as members of the group that represents the virtual host.

Defining ACLs

Users access resources in a WebLogic Server security realm. Whether or not a User can access a resource is determined by the access control lists ACLs for that resource. An ACL defines the permissions by which a User can interact with the resource. To define ACLs, you create an ACL for a resource, specify the permission for the resource and then grant the permission to a specified set of Users and Groups.

Each WebLogic Server resource has one or more permissions that can be granted. The following table summarizes the functions for various WebLogic Server resources for which permissions can be restricted with an ACL.

Table 12-18 ACLs for WebLogic Server Resources

For this WebLogic Server resource...	This ACL...	Grants Permission for these functions...
WebLogic Servers	`weblogic.server` `weblogic.server.`*`servername`*	boot **Note:** Only the `system` user can start a Managed Server.

Table 12-18 ACLs for WebLogic Server Resources

For this WebLogic Server resource...	This ACL...	Grants Permission for these functions...
Command-line Administration Tools	`weblogic.admin`	shutdown, lockserver unlockserver
WebLogic Events	`weblogic.servlet.`*`topicName`*	send receive
WebLogic servlets	`weblogic.servlet.`*`servletName`*	execute
WebLogic JDBC connection pools	`weblogic.jdbc.connectionPool.`*`poolname`*	reserve reset shrink
WebLogic Passwords	`weblogic.passwordpolicy`	unlockuser
WebLogic JMS destinations	`weblogic.jms.topic.`*`topicName`* `weblogic.jms.queue.`*`queueName`*	send, receive
WebLogic JNDI contexts	`weblogic.jndi.`*`path`*	lookup modify list

To create ACLs for a WebLogic Server resource, open the Administration Console and perform the following steps:

1. Specify the name of WebLogic Server resource that you want to protect with an ACL.

   For example, create an ACL for a JDBC connection pool named `demopool`.

2. Specify a permission for the resource.

   You can either create separate ACLs for each permission available for a resource or one ACL that grants all the permissions for a resource. For example, you can create three ACLs for the JDBC connection pool, `demopool`: one with `reserve` permission, one with `reset` permission, and one with `shrink` permission. Or you can create one ACL with `reserve`, `reset`, and `shrink` permissions.

3. Specify Users or Groups that have the specified permission to the resource.

When creating ACLs for resources in WebLogic Server you need to use the syntax in Table 12-18 to refer to the resource. For example, the JDBC connection pool named `demopool` would be specified as `weblogic.jdbc.connectionPool.demopool`.

Before you can boot a WebLogic Server, you need to give permission to the boot the server to a set of Users. This security measure prevents unauthorized Users from booting WebLogic Server.

Configuring the SSL Protocol

The Secure Sockets Layer (SSL) protocol provides secure connections by allowing two applications connecting over a network connection to authenticate the other's identity and by encrypting the data exchanged between the applications. The SSL protocol provides server authentication and optionally client authentication, confidentiality, and data integrity.

To configure the SSL protocol, perform the following steps:

1. Obtain a private key and digital certificate for WebLogic Server. You need a digital certificate and private key for each WebLogic Server that will use the SSL protocol.
2. Store the private key and digital certificate for WebLogic Server.
3. Through the Administration Console, set fields for the SSL protocol and the private key, digital certificate, and certificate authorities trusted by WebLogic Server. These fields are defined on a per-server basis; you must define them on any WebLogic Server that will use the SSL protocol.

The following sections describe these steps in detail.

For a complete description of the SSL Protocol, see Introduction to WebLogic Security and Security Fundamentals.

Requesting a Private Key and Digital Certificate

To acquire a digital certificate from a certificate authority, you must submit your request in a particular format called a Certificate Signature Request (CSR). WebLogic Server includes a Certificate Request Generator servlet that creates a CSR. The Certificate Request Generator servlet collects information from you and generates a private key file and a certificate request file. You can then submit the CSR to a certificate authority such as VeriSign or Entrust.net. Before you can use the Certificate Request Generator servlet, WebLogic Server must be installed and running.

To generate a CSR, perform the following steps:

1. Start the Certificate Request Generator servlet. The `.war` file for the servlet is located in the `\wlserver6.0\config\mydomain\applications` directory. The `.war` file is automatically installed when you start WebLogic Server.
2. In a Web browser, enter the URL for the Certificate Request Generator servlet as follows:

   `https://`*`hostname:port`*`/Certificate`

   The components of this URL are defined as follows:

   - *`hostname`* is the DNS name of the machine running WebLogic Server.
   - *`port`* is the number of the port at which WebLogic Server listens for SSL connections. The default is 7002.

     For example, if WebLogic Server is running on a machine named `ogre` and it is configured to listen for SSL communications at the default port `7002` to run the Certificate Request Generator servlet, you must enter the following URL in your Web browser:

     ```
     https://ogre:7002/certificate
     ```

3. The Certificate Request Generator servlet loads a form in your web browser. Complete the form displayed in your browser, using the information in the following table:

Table 12-19 Fields on the Certificate Request Generator Form

Field	Description
Country code	The two-letter ISO code for your country. The code for the United States is US.

Table 12-19 Fields on the Certificate Request Generator Form

Field	Description
Organizational unit name	The name of your division, department, or other operational unit of your organization.
Organization name	The name of your organization. The certificate authority may require any host names entered in this field belong to a domain registered to this organization.
E-mail address	The e-mail address of the administrator. The digital certificate is mail to this e-mail address.
Full host name	The fully-qualified name of the WebLogic Server on which the digital certificate will be installed. This name is the one used for DNS lookups of the WebLogic Server, for example, `node.mydomain.com`. Web browsers compare the host name in the URL to the name in the digital certificate. If you change the host name later, you must request a new digital certificate.
Locality name (city)	The name of your city or town. If you operate with a license granted by a city, this field is required; you must enter the name of the city that granted your license.
State name	The name of the State or Province in which your organization operates if your organization is in the United States or Canada, respectively. Do not abbreviate.
Private Key Password	The password used to encrypt the private key. Enter a password in this field if you want to use a protected key with WebLogic Server. If you choose to use a protected key, you are prompted for this password whenever the key is used. If you specify a password, you get a PKCS-8 encrypted private key. BEA recommends using a password to protect private keys. If you do not want to use a protected key, leave this field blank. To use protected private keys, enable the Use Encrytped Keys field on the SSL tab of the Server window in the Administration Console.

Table 12-19 Fields on the Certificate Request Generator Form

Field	Description
Random String	A string of characters to be used by the encryption algorithm. You do not have to remember this string in the future. It adds an external factor to the encryption algorithm, making it more difficult for anyone to break the encryption. For this reason, enter a string that is not likely to be guessed. BEA strongly recommends a long string with a good mixture of uppercase and lowercase letters, digits, spaces, and punctuation characters; these long, mixed strings contribute to more secure encryption.
Strength	The length (in bits) of the keys to be generated. The longer the key, the more difficult it is for someone to break the encryption. If you have the domestic version of WebLogic Server, you can choose 512-, 768-, or 1024-bit keys. We recommend the 1024-bit key.

4. Click the Generate Request button.

   The Certificate Request Generator servlet displays messages informing you if any required fields are empty or if any fields contain invalid values. Click the Back button in your browser and correct any errors.

   When all fields have been accepted, the Certificate Request Generator servlet generates the following files in the startup directory of your WebLogic Server:

   - `www_mydomain_com-key.der`—The private key file. The name of this file should go into the Server Key File Name field on the SSL tab in the Administration Console.
   - `www_mydomain_com-request.dem`—The certificate request file, in binary format.
   - `www_mydomain_com-request.pem`—The CSR file that you submit to the certificate authority. It contains the same data as the `.dem` file but is encoded in ASCII so that you can copy it into email or paste it into a Web form.

5. Select a certificate authority and follow the instructions on that authority's web site to purchase a digital certificate.

- VeriSign, Inc. offers two options for WebLogic Server: Global Site Services which features strong 128-bit encryption for domestic and export Web browsers, and Secure Site Services, which offers 128-bit encryption for domestic Web browsers and 40-bit encryption for export Web browsers.
- Entrust.net digital certificates offer 128-bit encryption for domestic browser versions and 40-bit encryption for export browser versions.

6. When you are instructed to select a server type, choose `BEA WebLogic Server` to ensure that you receive a digital certificate that is compatible with WebLogic Server.
7. When you receive your digital certificate from the certificate authority, you need to store it in the `\wlserver6.0\config\mydomain` directory.

Note: If you obtain a private key file from a source other than the Certificate Request Generator servlet, verify that the private key file is in PKCS#5/PKCS#8 PEM format.

8. Configure WebLogic Server to use the SSL protocol, you need to enter the following information on the SSL tab in the Server Configuration window:
   - In the Server Certificate File Name field, enter the full directory location and name of the digital certificate for WebLogic Server.
   - In the Trusted CA File Name field, enter the full directory location and name of the digital certificate for the certificate authority who signed the digital certificate of WebLogic Server.
   - In the Server Key File Name field, enter the full directory location and name of the private key file for WebLogic Server.

   For more information about configuring the SSL protocol, see Defining Fields for the SSL Protocol.
9. Use the following command-line option to start WebLogic Server.

   `-Dweblogic.management.pkpassword=`*`password`*

   where *`password`* is the password defined when requesting the digital certificate.

Storing Private Keys and Digital Certificates

Once you have a private key and digital certificate, copy the private key file generated by the Certificate Request Generator servlet and the digital certificate you received from the certificate authority into the `\wlserver6.0\config\mydomain` directory.

Private key files and digital certificates are generated in either PEM or Definite Encoding Rules (DER) format. The filename extension identifies the format of the digital certificate file.

A PEM (`.pem`) format private key file begins and ends with the following lines, respectively:

```
-----BEGIN ENCRYPTED PRIVATE KEY-----
```

```
-----END ENCRYPTED PRIVATE KEY-----
```

A PEM(`.pem`) format digital certificate begins and ends with the following lines, respectively:

```
-----BEGIN CERTIFICATE-----
```

```
-----END CERTIFICATE-----
```

Note: Your digital certificate may be one of several digital certificates in the file, each of which is bounded by the `BEGIN CERTIFICATE` and `END CERTIFICATE` lines. Typically, the digital certificate file for a WebLogic Server is in one file, with either a `.pem` or `.der` extension, and the WebLogic Server certificate chain is in another file. Two files are used because different WebLogic Servers may share the same certificate chain.

The first digital certificate in the certificate authority file is the first digital certificate in the WebLogic Server's certificate chain. The next certificates in the file are the next digital certificates in the certificate chain. The last certificate in the file is a self-signed digital certificate that ends the certificate chain.

A DER (`.der`) format file contains binary data. WebLogic Server requires that the file extension match the contents of the certificate file so be sure to save the file you receive from your certificate authority with the correct file extension.

Assign protections to the private key file and digital certificates so that only the `system` User of WebLogic Server has read privileges and all other users have no privileges to access the private key file or digital certificate. If you are creating a file

with the digital certificates of multiple certificate authorities or a file that contains a certificate chain, you must use PEM format. WebLogic Server provides a tool to for converting DER-format files to PEM format, and visa versa. For more information, see WebLogic Utilities.

Defining Trusted Certificate Authorities

When establishing an SSL connection, WebLogic Server checks the identity of the certificate authority against a list of trusted certificate authorities to ensure the certificate authority currently being used is trusted.

Copy the root certificate of the certificate authority into the `\wlserver6.0\config\mydomain` directory of your WebLogic Server and set the fields described in Defining Fields for the SSL Protocol.

If you want to use a certificate chain, append the additional PEM-encoded digital certificates to the digital certificate of the certificate authority that issued the digital certificate for WebLogic Server. The last digital certificate in the file should be a digital certificate that is self-signed (that is, the rootCA certificate).

If you want to use mutual authentication, take the root certificates for the certificate authorities you want to accept and include them to the trusted CA file.

Defining Fields for the SSL Protocol

To define fields for the SSL protocol, perform the following steps:

1. Open the Administration Console.
2. Open the Server Configuration window.
3. Select the SSL tab. Define the fields on this tab by entering values and checking the required checkboxes. (For details, see the following table.)
4. Click the Apply button to save your changes.
5. Reboot WebLogic Server.

The following table describes each field on the SSL tab of the Server Configuration window.

Note: Remember if you are using a PKCS-8 protected private key, you need to specify the password for the private key on the command line when you start WebLogic Server.

Table 12-20 SSL Protocol Fields

Field	Description
Enabled	Checkbox that enables the use of the SSL protocol. By default, this field is enabled.
SSL Listen Port	The number of the dedicated port on which WebLogic Server listens for SSL connections. The default is 7002.
Server Key File Name	The full directory location and name of the private key file for WebLogic Server. The file extension (`.DER` or `.PEM`) indicates the method that should be used by WebLogic Server to read the contents of the file.
Server Certificate File Name	The full directory location and name of the digital certificate file for WebLogic Server. The file extension (`.DER` or `.PEM`) indicates the method that should be used by WebLogic Server to read the contents of the file.
Server Certificate Chain File Name	The full directory location of the rest of the digital certificates for WebLogic Server. The file extension (`.DER` or `.PEM`) indicates the method that should be used by WebLogic Server to read the contents of the file.
Client Certificate Enforced	Checkbox that enables mutual authentication.
Trusted CA File Name	The name of the file that contains the digital certificate for the certificate authority(s) trusted by WebLogic Server. This file specified in this field can contain a single digital certificate or multiple digital certificates for certificate authorities. The file extension (`.DER` or `.PEM`) tells WebLogic Server how to read the contents of the file
CertAuthenticator	The name of the Java class that implements the CertAuthenticator interface. For more information about using the `weblogic.security.acl.CertAuthenticator` interface, see Mapping a Digital Certificate to a WebLogic User.

Table 12-20 SSL Protocol Fields

Field	Description
Use Java	Checkbox that enables the use of native Java libraries. WebLogic Server provides a pure-Java implementation of the SSL protocol: native Java libraries enhance the performance for SSL operations on the Solaris, Windows NT, and IBM AIX platforms. By default, this field is not enabled.
Use Encrypted Keys	Field that specifies that the private key for the WebLogic Server has been encyrpted with a password. The default is false.
Handler Enabled	Field that specifies whether or not WebLogic Server rejects SSL connections that fail client authentication for one of the following reasons: ■ The requested client digital certificate was not furnished. ■ The client did not submit a digital certificate ■ The digital certificate from the client was not issued by a certificate authority specified by the `Trusted CA Filename` field. By default, the SSL Handler allows one WebLogic Server to make outgoing SSL connections to another WebLogic Server. For example, an EJB in WebLogic Server may open an HTTPS stream on another Web server. With the `HandlerEnabled` field enabled, the WebLogic Server acts as a client in an SSL connection. By default this field is enabled. Disable this field only if you want to provide your own implementation for outgoing SSL connections. **Note:** The SSL Handler has no effect on the ability of WebLogic Server to manage incoming SSL connections.
Export Key Lifespan	The number of times WebLogic Server uses an exportable key between a domestic server and an exportable client before generating a new one. The more secure you want WebLogic Server to be the fewer times the key should be used before a new one is generated. The default is to use it 500 times.

Table 12-20 SSL Protocol Fields

Field	Description
Login Timeout Millis	The number of milliseconds that WebLogic Server should wait for an SSL connection before timing out. The default value is 25,000 milliseconds. SSL connections take longer to negotiate than regular connections. If clients are connecting over the Internet, raise the default number to accommodate additional network latency.
Certificate Cache Size	The number of digital certificates that are tokenized and stored by WebLogic Server. The default is 3. For more information, see Using Mutual Authentication with Applets.

Configuring Mutual Authentication

When WebLogic Server is configured for mutual authentication, clients are required to present their digital certificates to WebLogic Server which validates digital certificates against a list of trusted certificate authorities.

To configure your WebLogic Server for the SSL protocol and certificate authentication, complete the procedure in Configuring the SSL Protocol section.

Copy the root certificates for the certificate authorities to be used by WebLogic Server to the `\wlserver6.0\config\mydomain` directory. During mutual authentication, clients are required to present a digital certificate issued by one of these trusted certificate authorities.

To configure mutual authentication, check the Client Certificate Enforced option on the SSL tab in the Server Configuration window of the Administration Console. By default, this option is not enabled.

Configuring RMI over IIOP over SSL

The SSL protocol can be used to protect IIOP connections to RMI remote objects. The SSL protocol secures connections through authentication and encrypts the data exchanged between objects. To use the SSL protocol to protect IIOP over RMI connections, do the following:

1. Configure WebLogic Server to use the SSL protocol. For more information, see Configuring the SSL Protocol
2. Configure the client Object Request Broker (ORB) to use the SSL protocol. Refer to the product documentation for your client ORB for information about configuring the SSL protocol.
3. Use the `host2ior` utility to print the WebLogic Server IOR to the console. The `host2ior` utility prints two versions of the IOR, one for SSL connections and one for non-SSL connections. The header of the IOR specifies whether or not the IOR can be used for SSL connections.
4. Use the SSL IOR when obtaining the initial reference to the CosNaming service that accesses the WebLogic Server JNDI tree.

For more information about using RMI over IIOP, see Programming WebLogic IIOP over RMI.

Protecting Passwords

It is important to protect the passwords that are used to access resources in WebLogic Server. In the past, usernames and passwords were stored in clear text in a WebLogic Server security realm. Now WebLogic Server hashes all passwords. When WebLogic Server receives a client request, the password presented by the client is hashed and WebLogic Server compares it to the already hashed password for matching.

Each `filerealm.properties` file has an associated `SerializedSystemIni.dat` file that is used to hash the passwords. During installation, the `SerializedSystemIni.dat` file is put in the `\wlserver6.0\config\mydomain` directory. If for any reason the `SerializedSystemIni.dat` file is corrupted or destroyed, you must reconfigure WebLogic Server.

We recommend that you take the following steps:

- Make a backup copy of the `SerializedSystemIni.dat` file and put it in the same location as a copy of its associated `filerealm.properties` file.
- Set the permissions on the `SerializedSystemIni.dat` file protections such that the administrator of the WebLogic Server deployment has write and read privileges and no other users have any privileges.

If you already have a `weblogic.properties` file and you want to hash the passwords in the file, use the `Convert weblogic.properties` option on the main window in the Administration Console to convert the `weblogic.properties` file to a `config.xml` file. Once the file is converted, all existing passwords are protected.

Password guessing is a common type of security attack.In this type of attack, a hacker attempts to log in to a computer using various combinations of usernames and passwords. WebLogic Server has strengthened its protection against password guessing by providing a set of fields designed to protect passwords.

To protect the passwords in your WebLogic Server deployment, you must perform the following steps:

1. Open the Administration Console.
2. Open the Security Configuration window.
3. Select the Passwords tab. Define the desired fields on this tab by entering values at the appropriate prompts and checking the required checkboxes. (For details, see the following table).
4. Click the Apply button to save your choices.
5. Reboot WebLogic Server.

The following table describes each field on the Passwords tab of the Security Configuration window.

Table 12-21 Password Protection Fields

Field	Description
Minimum Password Length	Number of characters required in a password. Passwords must contain a minimum of 8 characters. The default is 8.
Lockout Enabled	Checkbox that requests the locking of a user account when an invalid attempt it made to log in to that account. By default, this field is enabled.
Lockout Threshold	The number of failed password entries for a user that can be tried to log in to a user account before that account is locked. Any subsequent attempts to access the account (even if the username/password combination is correct) raise a Security exception; the account remains locked until it is explicitly unlocked by the system administrator or another login attempt is made after the lockout duration period ends. Note that invalid login attempts must be made within a span defined by the `Lockout Reset Duration` field. The default is 5.
Lockout Duration	The number of minutes that a user's account remains inaccessible after being locked in response to several invalid login attempts within the amount of time specified by the `Lockout Reset Duration` field. The default is 30 minutes. In order to unlock a user account, you need to have the `unlockuser` permission for the `weblogic.passwordpolicy`.

Table 12-21 Password Protection Fields

Field	Description
Lockout Reset Duration	The number of minutes within which invalid login attempts must occur in order for the user's account to be locked. An account is locked if the number of invalid login attempts defined in the `Lockout Threshold` field happens within the amount of time defined by this field. For example, if the value in this field is five minutes and three invalid login attempts are made within a six-minute interval, then the account is not locked. If five invalid login attempts are made within a five-minute period, however, then the account is locked. The default is 5 minutes.
Lockout Cache Size	Specifies the intended cache size of unused and invalid login attempts. The default is 5.

Installing an Audit Provider

WebLogic Server allows you to create an audit provider to receive and process notifications of security events such as authentication requests, failed or successful authorization attempts, and receipt of invalid digital certificates.

To use an audit provider, you create an implementation of the `weblogic.security.audit.AuditProvider` interface. Then use the Administration Console to install and activate your implementation.

To install an audit provider, enter the name of your implementation of the `AuditProvider` class in the Audit Provider Class field on the Security Configuration window. Reboot WebLogic Server.

For more information about writing an audit provider, see Auditing Security Events. For an example of creating a connection filter, see the `LogAuditProvider` example in the `\samples\examples\security` directory of the WebLogic Server installation.

Installing a Connection Filter

You can create connection filters that allow you to reject or accept client connections based on a client's origin and protocol. After the client connects, and before any work is performed on its behalf, WebLogic Server passes the client's IP number and port, protocol (HTTP, HTTPS, T3, T3S, or IIOP), and WebLogic Server port number to the connection filter. By examining this information, you can choose to allow the connection or throw a `FilterException` to terminate it.

To use a connection filter, you must first create an implementation of the `weblogic.security.net.ConnectionFilter` interface. Then use the Administration Console to install your implementation.

To install a connection filter, enter the name of your implementation of the `weblogic.security.net.ConnectionFilter` interface, in the Connection Filter field on the General tab of the Security Configuration window in the Administration Console. Reboot WebLogic Server.

For information about writing a connection filter, see Filtering Network Connections. For an example of creating a connection filter, see the `SimpleConnectionFilter` example in the `\samples\examples\security` directory of the WebLogic Server installation..

Configuring Security Context Propagation

Security context propagation enables Java applications running in a WebLogic Server environment to access objects and operations in BEA WebLogic Enterprise (WLE) domains. The BEA WebLogic Enterprise Connectivity (WLEC) component of WebLogic Server provides the security context propagation capability.

When security context propagation is used, the security identity of a User defined in a WebLogic Server security realm is propagated as part of the service context of an Internet Inter-ORB Protocol (IIOP) request sent to the WLE domain over a network connection that is part of a WLEC connection pool. Each network connection in the WLEC connection pool has been authenticated using a defined User identity.

To use security context propagation, create a WLEC connection pool for each WLE domain you want to access from WebLogic Server. WebLogic Server populates each WLEC connection pool with IIOP connections. Java applications in a WebLogic Server environment obtain IIOP connections from a WLEC connection pool and use those connections to call objects and invoke operations in WLE domains.

Before using security context propagation, add *WLE_HOME*/lib/wleorb.jar and *WLE_HOME*/lib/wlepool.jar to the CLASSPATH variable in the startAdminWebLogic.sh or startAdminWebLogic.cmd file.

For more information, see Using WLEC.

The steps for implementing security context propagation are as follows:

1. Create a new WLEC connection pool for the purpose of security context propagation. To create a WLEC connection pool, go to the Services→WLEC node in the left pane of the Administration Console. In the right pane of the Administration Console, click the Create a new WLEC Connection Pool link. Define the fields in the following table:

Table 12-22 WLEC Connection Pool Fields on the General Tab

Field	Description
Name	The name of the WLEC connection pool. The name must be unique for each WLEC connection pool.

Primary Addresses	A list of addresses for IIOP Listener/Handlers that can be used to establish a connection between the WLEC connection pool and the WLE domain. The format of each address is *//hostname*:*port*.
	The addresses must match the ISL addresses defined in the UBBCONFIG file. Multiple addresses are seperated by semicolons. For example: //main1.com:1024; //main2.com:1044.
	To configure the WLEC connection pool to use the SSL protocol, use the corbalocs prefix with the address of the IIOP Listener/Handler. For example: corbalocs:*//hostname*:*port*.

Table 12-22 WLEC Connection Pool Fields on the General Tab

Field	Description
Failover Addresses	A list of addresses for IIOP Listener/Handlers that are used if connections cannot be established with the addresses defined in the Primary Addresses field. Multiple addresses are separated by semicolons. This field is optional.
Domain	The name of the WLE domain to which this WLEC connection pool connects. You can have only one WLEC connection pool per WLE domain. The domain name must match the `domainid` parameter in the `RESOURCES` section of the `UBBCONFIG` file for the WLE domain.
Minimum Pool Size	The number of IIOP connections to be added to the WLEC connection pool when WebLogic Server starts. The default is 1.
Maximum Pool Size	The maximum number of IIOP connections that can be made from the WLEC connection pool. The default is 1.

2. Click the Create button.
3. Propagate the security context for a User in a WebLogic Server security realm to a WLE domain. To do so, define the fields on the Security tab in the Connection Pool Configuration window. The following table describes these fields.

Table 12-23 WLEC Connection Pool Fields on the Security Tab

Field	Description
User Name	A WLE user name. This field is required only when the security level in the WLE domain is `USER_AUTH`, `ACL` or `MANDATORY_ACL`.
User Password	The password for the User defined in the `User Name` field. This field is required only when you define the `User Name` field.

Table 12-23 WLEC Connection Pool Fields on the Security Tab

Field	Description
User Role	The WLE user role. This field is required when the security level in the WLE domain is `APP_PW`, `USER_AUTH`, `ACL`, or `MANDATORY_ACL`.
Application Password	The WLE application password. This field is required when the security level in the WLE domain is `APP_PW`, `USER_AUTH`, `ACL`, or `MANDATORY_ACL`.
Minimum Encryption Level	The minimum SSL encryption level used between the WLE domain and WebLogic Server. The possible values are 0, 40, 56, and 128. The default is 40. Zero (0) indicates that the data is signed but not sealed. 40, 56, and 128 specify the length, in bits, of the encryption key. If this minimum level of encryption is not met, the SSL connection between WLE and WebLogic Server fails.
Maximum Encryption Level	The maximum SSL encryption level used between the WLE domain and WebLogic Server. The possible values are 0, 40, 56, and 128. The default is the maximum level allowed by the Encryption Package kit license. Zero (0) indicates that the data is signed but not sealed. 40, 56, and 128 specify the length, in bits, of the encryption key. If this minimum level of encryption is not met, the SSL connection between WLE and WebLogic Server fails.
Enable Certificate Authentication	Checkbox that enables the use of certificate authentication. By default certificate authentication is disabled.
Enable Security Context	Check this checkbox to pass the security context of the WebLogic Server User passed to the WLE domain. By default, security context is disabled.

4. To save your changes, click the Apply button and reboot WebLogic Server.

5. Run the `tpusradd` command to define the WebLogic Server User as an authorized User in the WebLogic Enterprise domain.

6. Set the `-E` option of the `ISL` command to configure the IIOP Listener/Handler to detect and utilize the propagated security context from the WebLogic Server realm. The `-E` option of the `ISL` command requires you to specify a principal name. The principal name defines the principal used by the WLEC connection pool to log in to the WebLogic Enterprise domain. The principal name should match the name defined in the User Name field when creating a WLEC connection pool.

Using certificate authentication between the WebLogic Server environment and the WebLogic Enterprise environment implies performing a new SSL handshake when establishing a connection from the WebLogic Server environment to a CORBA object, RMI object, or EJB in a WebLogic Enterprise environment is initiated. To support multiple client requests over the same SSL network connection, you must set up certificate authentication so that it operates as follows:

1. Obtain a digital certificate for the principal and put the private key in the `TUXDIR/udataobj/security/keys` directory of WebLogic Enterprise.

2. Use the `tpusradd` command to define the principal as a WebLogic Enterprise user.

3. Define the IIOP Listener/Handler in the `UBBCONFIG` file with the `-E` option to indicate the principal is to be used for authentication.

4. Define the principal name in the User Name field when creating a WLEC Connection pool in the Administration Console of WebLogic Server.

5. Obtain a digital certificate for the IIOP Listener/Handler.

6. Specify the digital certificate in the `SEC_PRINCIPAL_NAME` option of the ISL command and use the `-S` option to indicate that a secure port should be used for communication between the WebLogic Enterprise domain and the WebLogic Server security realm.

For more information about the `UBBCONFIG` file, see Creating a Configuration File in the WLE documentation.

For more information about the `corbalocs` prefix, see Understanding the Address Formats of the Bootstrap Object in the WLE documentation.

For information about WLE security levels, see Defining a Security Level in the WLE documentation.

Setting Up the Java Security Manager

When you run WebLogic Server under Java 2 (JDK 1.3), WebLogic Server uses the Java Security Manager to control access to WebLogic Server resources. The Java Security Manager requires a security policy file to set up the permissions. The WebLogic Server distribution includes a security policy file called `weblogic.policy` that contains a set of default permissions. With this file you can start WebLogic Server without first creating your own security policy file.

Edit the following lines in the `weblogic.policy` file, replacing the location of the directory in which you installed WebLogic Server.

```
grant codebase "file:./c:/weblogic/-"{
permission java.io.FilePermission "c:${/}weblogic${/}-", ...
```

Once you make these changes, we recommend that you take the following steps:

- Make a backup copy of the `weblogic.policy` file and put it in a secure place.
- Set the permissions on the `weblogic.policy` file protections such that the administrator of the WebLogic Server deployment has write and read privileges and no other users have any privileges.

Set the `java.security.manager` and `java.security.policy` properties on the Java command line when you start WebLogic Server. These properties perform the following functions:

- The `java.security.manager` property specifies that a security policy will be used by the Java Virtual Machine (JVM). You do not need to specify any arguments to this property.
- The `java.security.policy` property specifies the location of the security policy file to be used by the JVM. The argument to this property is the fully qualified file name of the `weblogic.policy` file.

For example:

```
$ java ... -Djava.security.manager\
-Djava.security.policy==c:/weblogic/weblogic.policy
```

Be sure to use == instead of = when specifying the `java.security.policy` argument so that only the `weblogic.policy` file is used by the Java security manager. The == causes the `weblogic.policy` file to override any default security policy. A single equal sign (=) causes the `weblogic.policy` file to be appended to an existing security policy.

Caution: The Java security manager is partially disabled during the booting of Administration and Managed Servers. During the boot sequence, the current Java security manager is disabled and replaced with a variation of the Java security manager that has the `checkRead()` method disabled. While disabling this method greatly improves the performance of the boot sequence, it also minimally diminishes security. The startup classes for WebLogic Server are run with this partially disabled Java security manager and therefore the classes need to be carefully scrutinized for security considerations involving the reading of files.

Modifying the weblogic.policy File for Third Party or User-Written Classes

The best location for your server-side user code is the `weblogic/myserver/serverclasses` directory. If you have third party or user-written classes that are not in that directory, perform the following steps to protect them:

1. Copy the entire block of code in the `weblogic.policy` file from `"grant codeBase..."` to the closing bracket and semicolon.
2. Paste the selection back into the `weblogic.policy` file below the section you just copied.
3. Edit the `grant codeBase` and the `permission.java.io.FilePermission` statements so that the directories point to the location of your third party or user-written code.

This procedure creates a security policy for your code that contains exactly the same permissions as those for WebLogic Server. You should examine these permissions closely to make sure that this is the security policy you want for those directories.

Caution: JavaSoft JDK version 1.2.1 on UNIX systems applies security policies improperly if your WebLogic Server software is not installed in the root directory of the file system or disk drive. Policy is only applied correct if the path in a `grant codeBase` URL has just one component. For example, if you install WebLogic Server in `c:\test\weblogic` (or even `/home/weblogic` on Solaris), you will see `AccessControlException` even though you use the correct URL in your security policy file.

To workaround this limitation, you can either install WebLogic in the root directory (recommended) or modify the URL so that it contains only the first component of the path to your WebLogic installation. For example:

```
grant codeBase "file:/c:/test/" {
```

Problems occur when using the "`/-`" in the specified URL. This problem has been acknowledged by Sun Microsystems as bug #4261298, but they have determined that this is not a bug in the JDK. They state, "when a path is trailed with "`/-`" it means that the element preceding it is a directory and that grant functions for all elements below it. It does not mean that you can read the directory itself." The workaround for this nuance is to add an additional `FilePermission` entry that consists of just the directory itself (with no trailing "`/-`').

CHAPTER

13 Managing Transactions

The following topics are discussed:

- Overview of Transaction Management
- Configuring Transactions
- Monitoring and Logging Transactions
- Moving a Server to Another Machine

This section provides 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" in the *Administration Guide*.

Overview of Transaction Management

The Administration Console provides an interface to the tools that allow you to enable and configure WebLogic Server features, including the JavaTransaction API (JTA). To invoke the Administration Console, see the procedures provided in Configuring WebLogic Servers and Clusters. The configuration process involves specifying values for attributes. These attributes define various aspects of the transaction environment, including the following:

- Transaction time-outs and limits
- Transaction manager behavior
- Transaction log file prefix

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 transactions 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

The Administration Console provides default values for all JTA configuration attributes. If you specify an invalid value for any configuration attribute, the WebLogic Server does not boot when you restart it.

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.

Once you have configured WebLogic JTA and any transaction participants, the system can perform transactions using the JTA API and the WebLogic JTA extensions.

You can configure any transaction attributes before running applications (static configuration) or, with one exception, at application run time (dynamic configuration). The `TransactionLogFilePrefix` attribute must be set before running applications.

To configure transaction attributes, complete the following procedure:

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. Click the JTA tab.
4. For each attribute, specify a value or, if available, accept the default value.
5. Click Apply to store new attribute values.
6. Ensure that the `Transaction Log File Prefix` attribute is set when you configure the server. For more information on setting the logging attribute, see "Monitoring and Logging Transactions.".

Table 13-1 briefly describes the transaction attributes available with WebLogic Server. For detailed information about attributes, and valid and default values for them, see the Domain topic in the Administration Console online help.

Table 13-1 Transaction Attributes

Attribute	Description
`Timeout Seconds`	The time, in seconds, a transaction may be active before the system forces a rollback.
`Abandon Timeout Seconds`	The maximum time, in seconds, that a transaction coordinator persists in attempting to complete a transaction.
`Before Completion Iteration Limit`	The number of `beforeCompletion` callbacks that are processed before the system forces a rollback.
`Max Transactions`	The maximum number of transactions that may be active on a particular server at one time.
`Max Unique Name Statistics`	The maximum number of unique transaction names that may be tracked by a server at one time.
`Forget Heuristics`	A Boolean value specifying whether the transaction manager should instruct a resource to forget any transaction with a heuristic outcome.

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, complete the following procedure:

1. Start the Administration Console.
2. Click the server node in the left pane.
3. Select a specific server in the left pane.
4. Click the Monitoring tab.
5. Click 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. Click the Logging tab.
7. Click the JTA tab.
8. Enter a transaction log file prefix then click on Apply to save the attribute setting.

For detailed information on monitoring and logging values and attributes, see the Server topic in the Administration Console online help.

Moving a Server to Another Machine

When an applications server is moved to another machine, it must be able to locate the transaction log files on the new disk. For this reason, we recommend moving the transaction log files to the new machine before starting the server there. By doing so

you can ensure that recovery runs properly. If the pathname is different on the new machine, update the `TransactionLogFilePrefix` attribute with the new path before starting the server.

When migrating 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. It is important to 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.

CHAPTER

14 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 (database connectivity with Java). For most JDBC administrative functions, which include creating, managing and monitoring connectivity, systems administrators use the Administrative Console or the command-line interface. Application developers may want to use the JDBC 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 Administrative Console

Your primary way to set and manage JDBC connectivity is through the Administration Console. Using the Administration Console, you set connectivity statically prior to starting the server. For more information, see “Starting the Administration Console” on page 1-3.

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. 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 programatically, see Programming WebLogic JDBC at http://e-docs.bea.com/wls/docs60/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 "Configuring WebLogic Servers and Clusters" on page 3-1.
- For a complete list of the JDBC attributes, see JDBC Connection Pool, JDBC Data Sources, JDBC MultiPools, and JDBC Transaction Data Sources in the WebLogic Administration Console Online Help at http://e-docs.bea.com/wls/docs60/ConsoleHelp/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 on the JDBC API, see *Programming WebLogic JDBC*. The "Introduction to WebLogic JDBC" section provides a concise overview of JDBC and JDBC drivers.
- For information on using the WebLogic jDrivers, see *Installing and Using WebLogic jDriver for Oracle* at http://e-docs.bea.com/wls/docs60/oracle/index.html, *Installing and Using WebLogic jDriver for Microsoft SQL Server* at http://e-docs.bea.com/wls/docs60/mssqlserver4/index.html, or *Installing and Using WebLogic jDriver for Informix* at http://e-docs.bea.com/wls/docs60/informix4/index.html.

Transactions (JTA)

- For information on managing JTA, see "Managing Transactions" on page 13-1.

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 on distributed transactions, see *Programming WebLogic JTA* at http://e-docs.bea.com/wls/docs60/jta/index.html.
- For information on using the WebLogic jDriver for Oracle/XA, see "Using WebLogic jDriver for Oracle/XA in Distributed Transactions" in *Installing and Using WebLogic jDriver for Oracle* at http://e-docs.bea.com/wls/docs60/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:

Connection Pools

A Connection Pool contains named groups of JDBC connections that are created when the Connection Pool is registered, usually when starting up WebLogic Server. 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/docs60/jdbc/programming.html#programming006.

All of the settings you make with the Administration Console are static; that is, all settings are made before WebLogic Server starts. You can create dynamic Connection Pools—after the server starts—using the command line (see “WebLogic Server Command-Line Interface Reference” on page B-1) or programatically using the API (see Creating a Dynamic Connection Pool in *Programming WebLogic JDBC*).

MultiPools

Used in local (non distributed) transactions on single-server WebLogic Server configurations, MultiPools aid in either:

- Load Balancing—pools are added without any attached ordering and are accessed using a round-robin scheme. When switching connections, the Connection Pool just after the last pool accessed is selected.
- High Availability—set up pools as an ordered list that determines the order in which Connection Pool switching occurs. For example, the first pool on the list is selected, then the second, etc.

All of the connections in a particular Connection Pool are identical; that is, they are attached to a single database. The Connection Pools within a MultiPool may, however, be associated with different DBMS. Read more about MultiPools in Programming WebLogic JDBC at http://e-docs.bea.com/wls/docs60/jdbc/programming.html#programming008.

Data Sources

A Data Source object enables JDBC clients to obtain a DBMS connection. Each Data Source object points to a Connection Pool or MultiPool. Data Source objects can be defined with or without JTA, which provides support for distributed transactions. Read more about Data Sources in Programming WebLogic JDBC at http://e-docs.bea.com/wls/docs60/jdbc/programming.html#programming001.

Note: Tx Data Sources cannot point to MultiPools, only Connection Pools, because MultiPools are not supported in distributed transactions.

JDBC Configuration Guidelines for Connection Pools, MultiPools and DataSources

This section describes JDBC configuration guidelines for 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 and, for dynamic connection pools, at the command line. There are three types of transactions:

- Local—non-distributed transaction
- Distributed with XA Driver—two-phase commit
- Distributed with non-XA Driver—single resource manager and single database instance

The following table describes how to use these objects in local and distributed transactions:

Table 14-1 Summary of JDBC Configuration Guidelines

Description/Object	Local Transactions	Distributed Transactions XA Driver	Distributed Transactions Non-XA Driver
JDBC driver	■ WebLogic jDriver for Oracle, Microsoft SQL Server, and Informix. ■ Compliant third-party drivers.	■ WebLogic jDriver for Oracle/XA. ■ Compliant third-party drivers.	■ WebLogic jDriver for Oracle, Microsoft SQL Server, and Informix. ■ Compliant third-party drivers.
Data Source	Data Source object recommended. (If there is no Data Source, use the JDBC API.)	Tx Data Source required.	Tx Data Source required. Set `enable two-phase commit=true` if more than one resource. See "Configuring Non-XA JDBC Drivers for Distributed Transactions" on page 14-17.
Connection Pool	Requires Data Source object when configuring in the Administration Console.	Requires TXData Source.	Requires TXData Source.

Table 14-1 Summary of JDBC Configuration Guidelines

Descriptio n/Object	Local Transactions	Distributed Transactions XA Driver	Distributed Transactions Non-XA Driver
MultiPool	Connection Pool and Data Source required. Used in single-server configurations only.	Not supported in distributed transactions.	Not supported in distributed transactions.

Note: Distributed transactions use the *WebLogic jDriver for Oracle/XA*, the transaction mode for WebLogic jDriver for Oracle.

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, Microsoft SQL Server, and Informix. 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

- Any JDBC 2.0 driver that supports JDBC 2.0 distributed transactions standard extension interfaces (`javax.sql.XADataSource`, `javax.sql.XAConnection`, `javax.transaction.xa.XAResource`), including the WebLogic jDriver for Oracle/XA.
- Any JDBC driver that supports JDBC 2.0 Core API but does not support JDBC 2.0 distributed transactions standard extension interfaces. Only one non-XA JDBC driver at a time can participate in a distributed transaction. See "Configuring Non-XA JDBC Drivers for Distributed Transactions" on page 14-17.

Configuring JDBC Drivers

This section explains how to configure drivers for local and distributed transactions.

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.

For more information on WebLogic two-tier JDBC drivers, refer to the BEA documentation for the specific driver you are using: *Installing and Using WebLogic jDriver for Oracle* at http://e-docs.bea.com/wls/docs60/oracle/index.html, *Installing and Using WebLogic jDriver for Microsoft SQL Server* at http://e-docs.bea.com/wls/docs60/mssqlserver4/index.html, or *Installing and Using WebLogic jDriver for Informix* at http://e-docs.bea.com/wls/docs60/informix4/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/docs60/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.

Table 14-2 WebLogic jDriver for Oracle: Connection Pool Configuration

Attribute **Name**	**Attribute Value**
Name	`myConnectionPool`
Targets	`myserver`
DriverClassname	`weblogic.jdbc.oci.Driver`
Initial Capacity	`0`
MaxCapacity	`5`
CapacityIncrement	`1`
Properties	`user=scott;server=localdb`

The following table shows a sample Data Source configuration using the WebLogic jDriver for Oracle.

Table 14-3 WebLogic jDriver for Oracle: Data Source Configuration

Attribute **Name**	**Attribute Value**
Name	`myDataSource`
Targets	`myserver`
JNDIName	`myconnection`
PoolName	`myConnectionPool`

The following table shows a sample Connection Pool configuration using the WebLogic jDriver for Microsoft SQL Server.

Table 14-4 WebLogic jDriver for Microsoft SQL Server: Connection Pool Configuration

Attribute **Name**	**Attribute Value**
Name	`myConnectionPool`
Targets	`myserver`
DriverClassname	`weblogic.jdbc.mssqlserver4.Driver`
Initial Capacity	`0`
MaxCapacity	`5`
CapacityIncrement	`1`
Properties	`user=sa;password=secret;db=pubs;server=myHost:1433;appname=MyApplication;hostname=myhostName`

The following table shows a sample Data Source configuration using the WebLogic jDriver for Microsoft SQL Server.

Table 14-5 WebLogic jDriver for Microsoft SQL Server: Data Source Configuration

Attribute **Name**	**Attribute Value**
Name	`myDataSource`
Targets	`myserver`
JNDIName	`myconnection`
PoolName	`myConnectionPool`

The following table shows a sample Connection Pool configuration using the WebLogic jDriver for Informix.

Table 14-6 WebLogic jDriver for Informix: Connection Pool Configuration

Attribute **Name**	**Attribute Value**
Name	`myConnectionPool`
Targets	`myserver`
DriverClassname	`weblogic.jdbc.informix4.Driver`
Initial Capacity	`0`
MaxCapacity	`5`
CapacityIncrement	`1`
Properties	`user=informix;password=secret;server=myDBHost;port=1493;db=myDB`

The following table shows a sample Data Source configuration using the WebLogic jDriver for Informix.

Table 14-7 WebLogic jDriver for Informix: Data Source Configuration

Attribute **Name**	**Attribute Value**
Name	`myDataSource`
Targets	`myserver`
JNDIName	`myconnection`
PoolName	`myConnectionPool`

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.

The following attributes are an example of a JDBC Connection Pool configuration using the WebLogic jDriver for Oracle in XA mode.

Table 14-8 WebLogic jDriver for Oracle/XA: Connection Pool Configuration

Attribute **Name**	**Attribute Value**
Name	`fundsXferAppPool`
Targets	`myserver`
DriverClassname	`weblogic.jdbc.oci.xa.XADataSource`
Initial Capacity	`0`
MaxCapacity	`5`

Table 14-8 WebLogic jDriver for Oracle/XA: Connection Pool Configuration

Attribute Name	Attribute Value
CapacityIncrement	`1`
Properties	`user=scott;password=tiger;server=localdb`

The following attributes are an example of a Tx Data Source configuration using the WebLogic jDriver for Oracle in XA mode.

Table 14-9 WebLogic jDriver for Oracle/XA: Tx Data Source

Attribute Name	Attribute Value
Name	`fundsXferData Source`
Targets	`myserver`
JNDIName	`myapp.fundsXfer`
PoolName	`fundsXferAppPool`

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 14-10 Oracle Thin Driver: Connection Pool Configuration

Attribute Name	Attribute Value
Name	`jtaXAPool`
Targets	`myserver,server1`
DriverClassname	`oracle.jdbc.xa.client.OracleXADataSource`
Initial Capacity	`1`

Table 14-10 Oracle Thin Driver: Connection Pool Configuration

Attribute **Name**	**Attribute Value**
MaxCapacity	`20`
CapacityIncrement	`2`
Properties	`user=scott;password=tiger; url=jdbc:oracle:thin:@baybridge:1521:bay817`

The following attributes are an example of a Tx Data Source configuration using the Oracle Thin Driver.

Table 14-11 Oracle Thin Driver: Tx Data Source Configuration

Attribute **Name**	**Attribute Value**
Name	`jtaXADS`
Targets	`myserver,server1`
JNDIName	`jtaXADS`
PoolName	`jtaXAPool`

Configure the JDBC Connection Pool for use with a Cloudscape driver as follows.

Table 14-12 Cloudscape: Connection Pool Configuration

Attribute **Name**	**Attribute Value**
Name	`jtaXAPool`
Targets	`myserver,server1`
DriverClassname	`COM.cloudscape.core.XADataSource`
Initial Capacity	`1`
MaxCapacity	`10`
CapacityIncrement	`2`

Table 14-12 Cloudscape: Connection Pool Configuration

Attribute **Name**	**Attribute Value**
Properties	`databaseName=CloudscapeDB`
SupportsLocalTransaction	`true`

Configure the Tx Data Source for use with a Cloudscape driver as follows.

Table 14-13 Cloudscape: Tx Data Source Configuration

Attribute **Name**	**Attribute Value**
Name	`jtaZADS`
Targets	`myserver,myserver1`
JNDIName	`JTAXADS`
PoolName	`jtaXAPool`

WebLogic jDriver for Oracle/XA Data Source Properties

Table 14-14 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 (Y) or a WebLogic Server extension to JDBC (N).

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 14-14. 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 14-14 Data Source Properties for WebLogic jDriver for Oracle/XA

Property Name	Type	Description	JDBC 2.0	Optional	Default Value
databaseName**	String	Name of a particular database on a server.	Y	Y	None
dataSourceName	String	A data source name; used to name an underlying XADataSource.	Y	Y	Connection Pool Name
description	String	Description of this data source.	Y	Y	None
networkProtocol**	String	Network protocol used to communicate with the server.	Y	Y	None
password	String	A database password.	Y	N*	None
portNumber**	Int	Port number at which a server is listening for requests.	Y	Y	None
roleName**	String	The initial SQL role name.	Y	Y	None

Table 14-14 Data Source Properties for WebLogic jDriver for Oracle/XA

Property Name	Type	Description	JDBC 2.0	Optional	Default Value
serverName	String	Database server name.	Y	Y*	None
user	String	User's account name.	Y	N*	None
openString	String	Oracle's XA open string.	N	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.	N	Y	true

Table 14-15 lists the mapping between Oracle's `xa_open` string fields and data source properties.

Table 14-15 Mapping of xa_open String Names to JDBC Data Source Properties

Oracle xa_open String Field Name	JDBC 2.0 Data Source Attribute	Optional
acc	user, password	N
sqlnet	ServerName	

Note also that users must specify `Threads=true` in Oracle's `xa_open` string. For complete description of Oracle's `xa_open` string fields, see your Oracle documentation.

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, specify the Enable Two-Phase Commit attribute 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 at a time 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 Enable Two-Phase Commit option unselected in the Console (accept the default enableTwoPhaseCommit = false). In this case, 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 Enable Two-Phase Commit in the Console (`enableTwoPhaseCommit = true`).

When `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 Enable Two-Phase Commit 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.

The following shows configuration attributes for a sample JDBC Connection Pool using a non-XA JDBC driver.

Table 14-16 WebLogic jDriver for Oracle: Connection Pool Configuration

Attribute **Name**	**Attribute Value**
Name	`fundsXferAppPool`
Targets	`myserver`
DriverClassname	`weblogic.jdbc.oci.Driver`
Initial Capacity	`0`
MaxCapacity	`5`
CapacityIncrement	`1`
Properties	`jdbc:weblogic:oracle`

The following table shows configuration attributes for a sample Tx Data Source using a non-XA JDBC driver.

Table 14-17 WebLogic j Driver for Oracle: Tx Data Source Configuration

Attribute **Name**	**Attribute Value**
Name	`fundsXferDataSource`
Targets	`myserver,server1`
JNDIName	`myapp.fundsXfer`
PoolName	`fundsXferAppPool`
EnableTwoPhaseCommit	`true`

Setting and Managing JDBC Connection Pools, MultiPools, and DataSources

The following sections discuss how to set database connectivity

- Statically, before the server starts, in the Administration Console, and
- Dynamically, after WebLogic Server has started, using the command-line interface.

Once connectivity is established, you use either the Administration Console or command-line interface to manage and monitor connectivity. See Table 14-19 for descriptions of the configuration tasks and links to the Administration Console Online Help.

JDBC Configuration and Assignment

Using the Administration Console, you statically set connectivity by specifying attributes and database properties for the JDBC components—Connection Pools, Data Sources, and MultiPools. See "Configuring JDBC Connectivity Using the Administration Console" on page 14-21.

Data Sources are associated with Connection Pools or MultiPools ("pool")—each Data Source is commonly associated with a specific pool. The associated Data Source and pool are then assigned to the same target—either the *same server* or *related server/cluster*. You cannot assign a Data Source to one server, then the Connection Pool to another.

Refer to the following table for more information.

Table 14-18 Configuration and Assignment Scenarios

Scenario #	Associate. . .	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 A and Connection Pool A to Managed Server 1. - AND - ■ Data Source a to Cluster X; then assign Connection Pool A to Managed Server 2 in Cluster X.	Data Source and Connection Pool assigned as a unit to two different targets.

(You can assign more than one Data Source to a pool, but there is no practical purpose for this.) You can assign these Data Source/pool combinations to more than one server or cluster, but they must be assigned in combination. For example, you can't assign a DataSource 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 command-line interface. See "JDBC Configuration Tasks Using the Command-Line Interface" on page 14-23. You can also configure dynamic Connection Pools programatically using the API (see Creating a Dynamic Connection Pool in *Programming WebLogic JDBC*).

JDBC Configurations for Servers or Clusters

Once you configure and associate the Data Source and Connection Pool (or MultiPool), you then assign each object to the same server or server/cluster. Some common scenarios are as follows:

- In a cluster, assign the Data Source to the cluster, and assign the associated Connection Pool to each managed server in the cluster.
- In a single server configuration, assign each Data Source and its associated Connection Pool to the server.
- 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 server.

See "Configuring JDBC Connectivity Using the Administration Console" on page 14-21 for a description of the tasks you perform.

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, complete the following procedure:

1. Start the Administration Console.
2. Locate the Services node in the left pane, then expand the JDBC node.
3. Select the tab 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 14-19.

The following table shows the connectivity tasks, listed in typical order in which you perform them. You may change the order; just remember you must configure an object before associating or assigning it.

Table 14-19 JDBC Configuration Tasks

Task #	JDBC Component/ Task	Description
1	Configure a Connection Pool	On the Configuration tabs, you set the attributes for the Connection Pool, such as Name, URL, and database Properties.
2	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.
3	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.
4	Configure a Data Source (and Associate with a 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.)
5	Configure a Tx Data Source (and Associate 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.
6	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 14-18 Configuration and Assignment Scenarios.
7	Assign the MultiPool to Servers or Clusters	Using the Target tab, you assign the configured MultiPool to Servers or Clusters.

JDBC Configuration Tasks Using the Command-Line Interface

The following table shows what methods you use to create a dynamic Connection Pool.

Table 14-20 Setting Connectivity—Dynamic

If you want to . . .	Then use the . . .
Create a dynamic Connection Pool	■ Command line—"CREATE_POOL" on page B-20, or ■ API—see "Configuring WebLogic JDBC Features" in *Programming WebLogic JDBC*

For more information, see "WebLogic Server Command-Line Interface Reference" on page B-1, and "Creating a Dynamic Connection Pool" 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 14-21 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.
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.

Table 14-21 JDBC Management Tasks

If you want to . . .	Do this . . . in the Administration Console
Delete a Connection Pool	See Delete a Connection Pool in the Online Help.
Delete a MultiPool	1. Select the MultiPools node in the left pane. The MultiPools table displays in the right pane showing all the MultiPools defined in your domain. 2. Click the Delete icon in the row of the MultiPool you want to delete. A dialog displays in the right pane asking you to confirm your deletion request. 3. Click Yes to delete the MultiPools. The MultiPool icon under the MultiPools node is deleted.
Delete a Data Source	1. Select the Data Sources node in the left pane. The Data Sources table displays in the right pane showing all the Data Sources defined in your domain. 2. Click the Delete icon in the row of the Data Source you want to delete. A dialog displays in the right pane asking you to confirm your deletion request. 3. Click Yes to delete the Data Source. The Data Source icon under the DataSources node is deleted.
Monitor a Connection Pool	1. Select the pool in the left pane. 2. Select the Monitoring tab in the right pane, then select the Monitor All Active Pools link.
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 (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 14-22 Managing Connection Pools with the Command-Line Interface

If you want to . . .	Then use this command . . .
Disable a Connection Pool	**"DISABLE_POOL" on page B-24**
Enable a Connection Pool that has been disabled	**"ENABLE_POOL" on page B-25**
Delete a Connection Pool	**"DESTROY_POOL" on page B-23**
Confirm if a Connection Pool was created	**"EXISTS_POOL" on page B-26**
Reset a Connection Pool	**"RESET_POOL" on page B-27**

CHAPTER

15 Managing JMS

This section explains how to manage WebLogic JMS, describing the following topics:

- Configuring JMS
- Monitoring JMS
- Recovering From a WebLogic Server Failure

Configuring JMS

Using the Administration Console, you define configuration attributes to:

- Enable JMS.
- Create JMS servers.
- Create and/or customize values for JMS servers, connection factories, destinations (queues and topics), destination templates, destination keys, backing stores, session pools, and connection consumers.
- Set up custom JMS applications.
- Define thresholds and quotas.
- Enable any desired JMS features, such as server clustering (see the next section), concurrent message processing, destination sort ordering, and persistent messaging.

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, the WebLogic Server will not boot JMS when you restart it. A sample JMS configuration is provided with the product.

When migrating from a previous release, the configuration information will be converted automatically, as described in "Migrating Existing Applications" in *Programming WebLogic JMS*.

To configure WebLogic JMS attributes, perform the following steps:

1. Start the Administration Console.
2. Select the JMS button under Services in the left pane to expand the list.
3. 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 begin sending and receiving messages using the JMS API. For more information about developing WebLogic JMS applications, see "Developing a WebLogic JMS Application" in *Programming WebLogic JMS*.

Note: To assist with your WebLogic JMS configuration planning, *Programming WebLogic JMS* provides configuration checklists that enable you to view the attribute requirements and/or options that support various JMS features.

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. You define and configure one or more connection factories to create connections with predefined attributes. 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 the cluster by configuring multiple connection factories and using *targets* to assign them to WebLogic Servers. Each connection factory can be deployed on multiple WebLogic Servers. For more information on JMS clustering, refer to "WebLogic JMS Fundamentals"in *Programming WebLogic JMS*.

To configure connection factories, use the Connection Factories node in the Administration Console to define the following:

- Configuration attributes including:
  - Name of the connection factory
  - Name for accessing the connection factory within the JNDI namespace
  - Client identifier (client ID) that can be used 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 (that is, priority, time-to-live, and 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.
  - Transaction attributes (transaction time-out and whether or not JTA user transactions are allowed)
- Targets (WebLogic Servers) that are associated with a connection factory to support clustering. Targets enable you to limit the set of servers, groups, and/or clusters on which a connection factory may be deployed.

WebLogic JMS defines one connection factory, by default: `weblogic.jms.ConnectionFactory`. All configuration attributes are set to their default values for this default connection factory, as described in "JMS Connection Factories" in the Administration Console Online Help.

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 JMS server on which the connection factory may be deployed. If you would like to target a particular JMS server, create a new connection factory and specify the appropriate JMS server target(s).

For instructions on creating and configuring a connection factory, see "JMS Connection Factories" 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 Templates

Templates provide an efficient way to define multiple destinations with similar attribute settings. 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 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 destination template configuration attributes, use the Templates node in the Administration Console. The configurable attributes for a destination 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 template specifies an override value for an attribute, the override value is used.

- The Name attribute is not inherited by the destination. This name is valid for the template only. All destinations must explicitly define a unique name.
- The JNDI Name, Enable Store, and Template attributes are not defined for destination templates.
- The Multicast attributes are not defined for destination 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, then you must be sure to specify a value within the destination 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 template, see "JMS Templates" in the Administration Console Online Help.

Configuring Destination Keys

Destination keys are used 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 "JMS Destination Keys" in the Administration Console Online Help.

Configuring Stores

The backing store consists of a file or database that is used for persistent messaging.

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 supports and provides JDBC drivers for the following databases:

- Cloudscape
- Informix
- Microsoft SQL (MSSQL) Server (Versions 6.5 and 7)
- Oracle (Version 8.1.6)
- Sybase (Version 12)

Optionally, you can restrict the access control list (ACL) for the JDBC connection pool. If you restrict this ACL, you must include the WebLogic *system* user and any user who sends JMS messages in the list. For more information, see "Managing Security" in the *Administration Guide.*

Note: The JMS examples are set up to work with the Cloudscape Java database. An evaluation version of Cloudscape is included with WebLogic Server and a *demoPool* database is provided.

To create a file or database store, use the Stores node in the Administration Console and define the following configuration attributes:

- Name of the backing store
- (For a file store) Directory within which file stores will be saved
- (For a JDBC database store) JDBC connection pool and database table name prefix for use with multiple instances

Warning: You cannot configure a XA connection pool with a JDBC database store.

Note: JMS backing stores can increase the amount of memory required during initialization of a WebLogic Server as the number of stored messages increases. If initialization fails due to insufficient memory when you are

rebooting a 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 backing store. Try rebooting again.

About JMS Stores

The JMS database contains two system tables that are generated automatically and are used internally by JMS, as follows:

- `<prefix>JMSStore`
- `<prefix>JMSState`

The prefix name uniquely identifies JMS tables in the backing store. Specifying unique prefixes allows multiple stores to exist in the same database. The prefix is configured via the Administration Console when configuring the 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.

The prefix should be specified using the following format, which will result in a valid table name when prepended to the JMS table name:

```
[[catalog.]schema.]prefix
```

Warning: No two JMS stores should be allowed to use the same database tables, as this will result in data corruption.

For instructions on creating and configuring a store, see "JMS File Stores" and "JMS JDBC Stores" for information about file and JDBC database stores, respectively, in the Administration Console Online Help.

Recommended JDBC Connection Pool Settings for JMS Stores

For implementations using a JDBC store, if the DBMS should shut down and then come back online, WebLogic JMS will not be able to access the store until WebLogic Server is shutdown and restarted. To avoid this situation, you should configure the following attributes on the JDBC Connection Pool associated with the JMS store:

```
TestConnectionsOnReserve="true"
TestTableName="[[[catalog.]schema.]prefix]JMSState"
```

Otherwise, if the JDBC resource goes down and comes back up, JMS cannot re-use it until the WebLogic Server is shutdown and restarted.

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:

- Configuration attributes including:
  - Name of the JMS server.
  - Backing store (file or JDBC database) required for persistent messaging. If you do not assign a backing store for a JMS server, persistent messaging is not supported on that server.
  - Template that is used to create all temporary destinations, including temporary queues and temporary topics.
  - Thresholds and quotas for messages and bytes (maximum number, and high and low thresholds).
- Targets (WebLogic Servers) that are associated with a JMS server to support clustering. Targets enable you to limit the set of servers, groups, and/or clusters on which the JMS server may be deployed.

Note: The deployment of a JMS server differs from that of a connection factory or template. A JMS server is deployed on a single server. A connection factory or template can be instantiated on multiple servers simultaneously.

For instructions on creating and configuring a JMS server, see "JMS Servers" in the Administration Console Online Help.

Configuring Destinations

A destination identifies a queue or topic. Once you have defined a JMS server, you can configure its destinations. You can configure one or more destinations for each JMS server.

You can 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 Templates" on page 15-4.

To configure destinations explicitly, use the Destinations node in the Administration Console, and define the following configuration attributes:

- 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
- Template used for creating destinations
- Keys used to define the sort order for a specific destination
- Thresholds and quotas for messages and bytes (maximum number, and high and low thresholds)
- Message attributes that can be overridden (such as priority, time-to-live, and delivery mode)
- Multicasting attributes, including multicast address, port, and time-to-live (for topics only)

For instructions on creating and configuring a destination, see "JMS Destinations" 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 Session Pools

Server session pools enable an application to process messages concurrently. Once you have defined a JMS server, you have the option of configuring its server session pools. You can configure one or more session pools for each JMS server.

To configure server session pools, use the Session Pools node in the Administration Console and define the following configuration attributes:

- Name of the server session pool
- 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 "JMS Session Pools" 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 retrieve server sessions and process messages. Once you have defined a session pool, you have the option of configuring its connection consumers. You can 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
- 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 "JMS Connection Consumers" in the Administration Console Online Help.

Monitoring JMS

Statistics are provided for the following JMS objects: JMS servers, connections, sessions, destinations, message producers, message consumers, and server session pools. You can monitor JMS statistics using the Administration Console.

JMS statistics continue to increment as long as the server is running. Statistics can be reset only when the server is rebooted.

To view JMS monitoring information, perform the following steps:

1. Start the Administration Console.
2. Select the JMS node under Services, in the left pane, to expand the list of JMS services.

3. Select the JMS Server node under JMS in the left pane.

   The JMS Servers information is displayed in the right pane.

4. Select the JMS server that you want to monitor from the JMS server list or, from the JMS Servers displayed in the right pane.

   The specified JMS server information is displayed in the right pane.

5. Select the Monitoring tab to display the monitoring data.

For detailed information about the information being monitored, see the Administration Console Online Help.

Recovering From a WebLogic Server Failure

The following sections describe how to restart or replace a WebLogic Server in the event of a system failure, and provide programming considerations for gracefully terminating an application following such an event.

Restarting or Replacing the WebLogic Server

In the event that a WebLogic Server fails, there are three methods that you can use to perform a system recovery:

- Restart the failed server
- Start up a new server using the same IP address as the failed server
- Start up a new server using a different IP address than the failed server

To restart the failed server or start up a new server using the same IP address as the failed server, boot the server and start the server processes, as described in Starting and Stopping WebLogic Servers in the *Administration Guide*.

To start up a new server using a different IP address than the failed server:

1. Update the Domain Name Service (DNS) so that the server alias references the new IP address.

2. Boot the server and start the server processes, as described in Starting and Stopping WebLogic Servers in the *Administration Guide*.
3. Optionally perform one or more of the following tasks

If your 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.
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 re-started at its new location. Otherwise, transactions in the process of being committed at that time of the crash might not be resolved correctly, resulting in data inconsistencies. All uncommitted transactions will be rolled back.

Note: JMS backing stores can increase the amount of memory required during initialization of a WebLogic Server as the number of stored messages increases. When rebooting a 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 backing store and try the reboot again.

Programming Considerations

You may want to program your application to terminate gracefully in the event of a WebLogic Server failure. For example:

If a WebLogic Server Fails and...	Then...
You are connected to the failed WebLogic Server	A `JMSException` will be 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	You must re-establish everything once the server is restarted or replaced.
A JMS Server is targeted on the failed WebLogic Server	A `ConsumerClosedException` will be delivered to the session exception listener. You must re-establish any message consumers that may have been lost.

CHAPTER

16 Managing JNDI

This section discusses the following topics:

- Loading Objects in the JNDI Tree
- Viewing the JNDI Tree

Loading Objects in the JNDI Tree

Before you can access an object using WebLogic JNDI, you must load the object in the WebLogic Server JNDI tree. The Administration Console in WebLogic Server allows you to load J2EE services in WebLogic Server in the JNDI tree. WebLogic Server EJBs, RMI, JMS, JDBC, and Mail objects can be loaded in the JNDI tree.

To load an object in the JNDI tree, choose a name under which you want the object to be loaded to the JNDI tree and enter it in the JNDI Name field when you create the object.

Viewing the JNDI Tree

From time to time you may find it useful to view the objects in the WebLogic Server JNDI tree. To view the JNDI tree, click the View JNDI Tree link on the Monitoring tab for your WebLogic Server deployment.

CHAPTER

17 Managing WebLogic Server Licenses

Your WebLogic Server requires a valid license to run. The following sections tells you how to install and update WebLogic licenses:

- Installing a WebLogic License
- Updating a License

Installing a WebLogic License

Using Evaluation Licenses

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 3 unique client IP addresses.

If you downloaded WebLogic Server from the BEA website, 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 30-day evaluation license.

In either 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 java (Java 2) is in your path. To add the JDK to your path, enter one of the following commands:
   - `set PATH=.\jdk130\bin;%PATH%` (Windows systems)
   - `set PATH=./jdk130/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 Server Java Utilities

WebLogic Server provides several Java programs that simplify installation and configuration tasks, provide services, and offer convenient shortcuts. This section describes each Java utility provided with WebLogic Server. The command-line syntax is specified for all utilities and, for some, examples are provided. The following utilities are documented:

- `AppletArchiver`
- `Conversion`
- `der2pem`
- `dbping`
- `deploy`
- `getProperty`
- `logToZip`
- `MulticastTest`
- `myip`
- `pem2der`
- `Schema`
- `showLicenses`
- `system`
- `t3dbping`
- `verboseToZip`
- `version`

- `writeLicense`

To use these utilities you must correctly set your `CLASSPATH`. For more information, see "Setting the Classpath Option."

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

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 Server, 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 file name 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 two-tier WebLogic jDriver.

Syntax

```
$ java -Dbea.home=WebLogicHome utils.dbping DBMS user password DB
```

Argument	Definition
WebLogicHome	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`
	WebLogic jDriver for Informix:
	`INFORMIX4`
	Oracle Thin Driver:
	`ORACLE_THIN`
	Sybase JConnect driver:
	`JCONNECT`

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* WebLogic jDriver for Informix: *DBNAME*@*HOST*:*PORT* Oracle Thin Driver: *HOST*:*PORT*:*DBNAME* Sybase JConnect driver: JCONNECT: *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.)s

deploy

The `deploy` utility gets a J2EE application from an archive file (`.jar`, `.war`, or `.ear`) and deploys the J2EE application to a running WebLogic Server. For additional information, see Deploying and Configuring Web Applications in the *WebLogic Server Administration Guide*, and the programming guide, Developing WebLogic Server Applications.

Syntax

```
$ java weblogic.deploy [options] [list|deploy|undeploy|update]
   password {application} {source}
```

Arguments

Argument	Description
`applications`	Required. Idenitfies the name of the application. The application name can be specified at deployment time, either with the deployment or console utilities.
`deploy`	Optional. Deploys a J2EE application `.jar`, `.war`, or `.ear` file to the specified server.
`list`	Optional. Lists all applications in the specified WebLogic Server.
`password`	Required. Specifies the system password for the WebLogic Server.
`source`	Required. Specifies the exact location of the application archive file (`.jar`, `.war`, or `.ear`), or the path to the top level of an application directory.
`undeploy`	Optional. Removes an existing application from the specified server.
`update`	Optional. Re-deploys an application in the specified server.

Options

Option	Definition
`-component` *`componentname:target1, target2`*	Component to be deployed on various targets, must be specified as: *`componentname:target1,target2`* where *`componentname`* is the name of the `.jar` or `.war` file without the extension. This option can be specified multiple times for any number of components (`.jar` or `.war`). An `.ear` file cannot be deployed. Each of its components must be deployed separately using this option.
`-debug`	Prints detailed debugging information to `stdout` during the deployment process.
`-help`	Prints a list of all options available for the deploy utility.
`-host` *`host`*	Specifies the host name of the WebLogic Server to use for deploying the J2EE application (`.jar`, `.war`, `.ear`). If you do not specify this option, the deploy utility attempts to connect using the host name `localhost`.
`-port` *`port`*	Specifies the port number of the WebLogic Server to use for deploying the J2EE application .`jar`, .`war`, or .`ear` file. **Note:** If you do not specify the `-port` option, `deploy` connects uses a default of 7001.
`-url` *`url`*	Specifies the URL of a Weblogic Server. The default is `localhost:7001`.
`-username` *`username`*	Name of the user with which a connection will be made. The default is `system`.
`-version`	Prints the version of the deploy utility.

Examples

The deploy utility is useful for various purposes, including the following:

- Viewing a Deployed J2EE Application
- Deploying a New J2EE Application
- Removing a Deployed J2EE Application
- Updating a Deployed J2EE Application

Viewing a Deployed J2EE Application

To view an application that is deployed on a local WebLogic Server, enter the following command:

```
% java weblogic.deploy list password
```

The value of *password* is the password for the WebLogic Server system account.

To list a deployed application on a remote server, specify the `port` and `host` options, as follows:

```
% java weblogic.deploy -port port_number -host host_name list password
```

Deploying a New J2EE Application

To deploy a J2EE application file (`.jar`, `.war`, or `.ear`) or application directory that is not deployed to WebLogic Server, enter the following command:

```
% java weblogic.deploy -port port_number -host host_name
   deploy password application source
```

The values are as follows:

- *application* is the string you want to assign to this Application.
- *source* is the full pathname of the J2EE application file (`.jar`,`.war`,`.ear`) you want to deploy, or the full pathname of the application directory.

For example:

```
% java weblogic.deploy -port 7001 -host localhost deploy weblogicpwd Basic_example
   c:\mysamples\ejb\basic\BasicStatefulTraderBean.jar
```

Note: The J2EE application file (`.jar`,`.war`,`.ear`) copied to the applications directory of the Administration Server is renamed with the name of the application. Therefore, in the previous example, the name of the application archive `. . ./config/mydomain/applications` directory is changed from `BasicStatefulTraderBean.jar` to `Basic_example.jar`.

Removing a Deployed J2EE Application

To remove a deployed J2EE application, you need only reference the assigned application name, as shown in the following example:

```
% java weblogic.deploy -port 7001 -host localhost undeploy
  weblogicpwd Basic_example
```

Note: Removing a J2EE application does not remove the application from WebLogic Server. You cannot re-use the application name with the `deploy` utility. You can re-use the application name to `update` the deployment, as described in the following section.

Updating a Deployed J2EE Application

To update a J2EE application, use the `update` argument and specify the name of the active J2EE application as follows:

```
% java weblogic.deploy -port 7001 -host localhost update
   weblogicpwd Basic_example
   c:\updatesample\ejb\basic\BasicStatefulTraderBean.jar
```

To update a specific component on one or more servers, enter the following command:

```
% java weblogic.deploy -port 7001 -host localhost –component
   BasicStatefulTraderBean.jar:sampleserver,exampleserver update
   weblogicpwd Basic_example
   c:\updatesample\ejb\basic\BasicStatefulTraderBean.jar
```

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:\
```

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 out:

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 ServerClusters*.

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 when the -verbose argument is used. Text file with SQL statements.

Example

The following code shows a sample Schema command line:

```
$ java utils.Schema "jdbc:cloudscape:demo;create=true"
   COM.cloudscape.core.JDBCDriver
   -verbose examples/utils/ddl/demo.ddl
```

The following code shows a sample .ddl file:

```
DROP TABLE ejbAccounts;
CREATE TABLE ejbAccounts
  (id   varchar(15),
   bal  float,
   type varchar(15));
DROP TABLE idGenerator;
CREATE TABLE idGenerator
```

```
(tablename varchar(32),
 maxkey    int);
```

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
```

t3dbping

The `t3dbping` utility tests a WebLogic JDBC connection to a DBMS via any two-tier JDBC driver. You must have access to a WebLogic Server and a DBMS to use this utility.

Syntax

```
$ java utils.t3dbping WebLogicURL username password DBMS
driverClass driverURL
```

Argument	Definition
WebLogicURL	Required. URL of the WebLogic Server.
username	Required. Valid username of DBMS user.
password	Required. Valid password of DBMS user.
DBMS	Required. Database name.
driverClass	Required. Full package name of the WebLogic Server two-tier driver.
driverURL	Required. URL of the WebLogic Server two-tier driver.

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 Server via `stdout`.

Syntax

```
$ java weblogic.version
```

Example

```
$ java weblogic.version
WebLogic Build: 4.0.1 04/05/1999 22:02:11 #41864
```

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 Server installation). **Note:** 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 describe the WebLogic Server command-line interface:

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

The examples in this document are based on the following assumptions:

- WebLogic Server is installed in the `c:/weblogic` directory.
- The JDK is located in the `c:/java` directory.
- You have started WebLogic Server from the directory in which it was installed.

Before you can run WebLogic Server commands, you must do the following:

1. Install and configure the WebLogic Server software, as described in the *WebLogic Server Installation Guide*. See `http://e-docs.bea.com/wls/docs60/install/index.html`.
2. Set CLASSPATH correctly. See "Setting the Classpath Option" at `http://e-docs.bea.com/wls/docs60/adminguide/startstop.html#SettingClasspath`.
3. Enable the command-line interface by performing one of the following steps:
   - Start the server from the directory in which it was installed.
   - If you are not starting the server from its installation directory, enter the following command, replacing `c:/weblogic` with the name of the directory in which the WebLogic Server software is installed:

     `-Dweblogic.system.home=c:/weblogic`

Using WebLogic Server Commands

This section presents the syntax and required arguments for using WebLogic Server commands. WebLogic Server commands are not case-sensitive.

Syntax

```
java weblogic.Admin [–url URL] [-username username]
   [-password password] COMMAND arguments
```

Arguments

The following arguments are required by many WebLogic Server commands.

Argument	Definition
URL	The URL of the WebLogic Server host including the number of the TCP port at which WebLogic Server is listening for client requests. The format is *hostname*:*port*. The default is `localhost:7001`. **Note:** The URL used with a server command always refers to the WebLogic Server, while the URL used with run-time and configuration Mbean commands always refers to a specific Administration server.
username	Optional. Username to be authenticated so commands can be executed. Default is `guest`.
password	Optional. Password to be authenticated so commands can be executed. Default is `guest`.

An administrator must have the appropriate access control permissions to run commands used to manage run-time Mbeans.

See the following sections:

- "WebLogic Server Administration Command Reference" on page B-3
- "Mbean Management Command Reference" on page B-28

WebLogic Server Administration Command Reference

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.

Table B-1 WebLogic Server Administration Commands Overview

Task	Command	Description
Cancel shut down a WebLogic Server	`CANCEL_SHUTDOWN`	Cancels the SHUTDOWN command for the WebLogic Server that is specified in the URL. See "CANCEL_SHUTDOWN" on page B-6.
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-7.
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-8.
View WebLogic Server licenses	`LICENSES`	Lists the licenses for all the WebLogic Server instances installed on a specific server. See "LICENSES" on page B-9.
List JNDI naming tree node bindings	`LIST`	Lists the bindings of a node in the JNDI naming tree. See "LIST" on page B-10.
Lock WebLogic Server	`LOCK`	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-11.
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-12.
Viewing server log files	`SERVERLOG`	Displays the server log file generated on a specific server. See "SERVERLOG" on page B-13.
Shut down a WebLogic Server	`SHUTDOWN`	Shuts down the WebLogic Server that is specified in the URL. See "SHUTDOWN" on page B-14.
View threads	`THREAD_DUMP`	Provides a real-time snapshot of the WebLogic Server threads that are currently running. See "THREAD_DUMP" on page B-15.

Table B-1 WebLogic Server Administration Commands Overview (Continued)

Task	Command	Description
Unlock a WebLogic Server	`UNLOCK`	Unlocks the specified WebLogic Server after a `LOCK` operation. See "UNLOCK" on page B-16.
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-17.

CANCEL_SHUTDOWN

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-14 and "UNLOCK" on page B-16.

Syntax

```
java weblogic.Admin [-url URL] [-username username]
   [-password password] CANCEL_SHUTDOWN
```

Example

In the following example, a system user named `system` with a password of `gumby1234` requests to cancel the shutdown of the WebLogic Server listening on port 7001 on machine `localhost`:

```
java weblogic.Admin -url t3://localhost:7001 -username system
   -password gumby1234 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 [-url URL] [-username username]
   [-password password] 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
```

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 [-url URL] [-username username]
   [-password password] LICENSES
```

Example

In the following example, an administrator using the default username (`guest`) and default password (`guest`) requests the license information for a WebLogic Server running on port 7001 of machine `localhost`:

```
java weblogic.Admin -url localhost:7001 -username guest
   -password guest LICENSES
```

LIST

Lists the bindings of a node in the JNDI naming tree.

Syntax

```
java weblogic.Admin [-username username] [-password password]
   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 -username adminuser -password gumby1234
   LIST weblogic.ejb
```

LOCK

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.

Syntax

```
java weblogic.Admin [-url URL] [-username username]
   [-password password] 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`.

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 [-url URL] [-username username]
   [-password password] 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
```

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 [-url URL] [-username username]
   [-password password] 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
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 WebLogic Server that is specified in the URL.

Syntax

```
java weblogic.Admin [-url URL] [-username username]
   [-password password] SHUTDOWN [seconds] [“lockMessage”]
```

Argument	Definition
seconds	Optional. Number of seconds allowed to elapse between the invoking of this command and the shutdown of the server.
“lockMessage”	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 an administrative password of `gumby1234` shuts down a WebLogic Server that is listening on port 7001 of machine `localhost`:

```
java weblogic.Admin -url localhost:7001 -username adminuser
-password gumby1234 SHUTDOWN 300 “Server localhost is shutting
down.”
```

After the command is issued, an interval of five minutes (300 seconds) elapses. Then the command shuts down the specified server and sends the following message to `stdout`:

```
Server localhost is shutting down.
```

THREAD_DUMP

Provides a real-time snapshot of the WebLogic Server threads that are currently running.

Syntax

```
java weblogic.Admin [-url URL] [-username username]
   [-password password] THREAD_DUMP
```

UNLOCK

Unlocks the specified WebLogic Server after a `LOCK` operation.

Syntax

```
java weblogic.Admin [-url URL] [-username username]
   [-password password] 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 -url URL -username username
   -password password 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 guest
   -password guest VERSION
```

Note: In this example, the default value of both the *username* and *password* arguments, `guest`, 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/docs60/jdbc/index.html` and Managing JDBC Connectivity in the *Administration Guide* at `http://e-docs.bea.com/wls/docs60/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-20
Destroy a Connection Pool	`DESTROY_POOL`	Connections are closed and removed from the pool and the pool dies when it has no remaining connections. Only the "system" user or users granted "admin" permission by an ACL associated with a connection pool can destroy the pool. See **"DESTROY_POOL" on page B-23**.
Disable a Connection Pool	`DISABLE_POOL`	You can temporarily disable a connection pool, preventing any clients from obtaining a connection from the pool. Only the "system" user or users granted "admin" permission by an ACL associated with a connection pool can disable or enable the pool. See **"DISABLE_POOL" on page B-24**.
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-25**.

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-26**.
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-27**.

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/docs60/jdbc/programming.html#dynamic_conn_pool.

Syntax

```
java weblogic.Admin [-url URL] [-username username]
   [-password password] 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 `adminuser` and the password `gumby1234` runs the `CREATE_POOL` command to create a dynamic connection pool:

```
java weblogic.Admin -url localhost:7001 -username adminuser
   -password gumby1234 CREATE_POOL myPool

java weblogic.Admin -url t3://forest:7901 -username system
   -password gumby1234 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. Only the "system" user or users granted "admin" permission by an ACL associated with a connection pool can destroy the pool.

Syntax

```
java weblogic.Admin [-url URL] [-username username]
   [-password password] 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. Only the "system" user or users granted "admin" permission by an ACL associated with a connection pool can disable or enable 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 [-url URL] [-username username]
   [-password password] 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 [-url URL] [-username username]
   [-password password] 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 [-url URL] [-username username]
   [-password password] 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 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 URL RESET_POOL poolName system password
```

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."`
poolName	Name of a connection pool as it is registered in the WebLogic Server's config.xml file.
password	Administrative password for the user "system". You must supply the username "system" and the administrative password to use this Admin command.

Example

This command refreshes the connection pool registered as "eng" for the WebLogic Server listening on port 7001 of the host xyz.com.

```
java weblogic.Admin t3://xyz.com:7001 RESET_POOL eng system gumby
```

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 configuration Mbeans	`CREATE`	Creates an instance of a configuration Mbean. Returns `OK` to `stdout` when successful. This command cannot be used for run-time Mbeans. See "CREATE" on page B-29.
Delete configuration Mbeans	`DELETE`	Deletes a configuration Mbean. Returns `OK` in `stdout` when successful. This command cannot be used for run-time Mbeans. See "DELETE" on page B-30.
View run-time Mbean attributes	`GET`	Displays run-time Mbean attributes. See "GET" on page B-31.
Invoke run-time Mbeans	`INVOKE`	Invokes methods that are not designed to get or set attributes. This command can call only run-time Mbeans. See "INVOKE" on page B-33.
View run-time metrics and statistics	`INVOKE` `GET`	Run the `INVOKE` and `GET` commands to view run-time metrics and statistics. These commands can call only run-time Mbeans. See "INVOKE" on page B-33, and "GET" on page B-31.
Set configuration Mbean attributes	`SET`	Sets the specified attribute values for the named configuration Mbean. Returns `OK` on `stdout` when successful. This command cannot be used for run-time Mbeans. See "SET" on page B-34.

CREATE

Creates an instance of a configuration Mbean. Returns `OK` to `stdout` when successful. This command cannot be used for run-time Mbeans. The Mbean instance is saved in the `config.xml` file or the security realm, depending on where the changes have been made.

Note: When you create Mbeans, configuration objects are also created.

For more information about creating Mbeans, see *Developing WebLogic Server Applications,* at `http://e-docs.bea.com/wls/docs60/programming/index.html`.

Syntax

```
java weblogic.Admin [–url URL] [-username username]
   [-password password] CREATE –name name –type mbean_type
   [–domain domain_name]

java weblogic.Admin [–url URL] [-username username]
   [-password password] CREATE –mbean mbean_name
```

Argument	Definition
name	Required. The name you choose for the Mbean that you are creating.
mbean_type	Required. When creating attributes for multiple objects of the same type.
mbean_name	Required. Fully qualified name of an Mbean, in the following format: "*domain*:Type=*type*,Name=*name*" `Type` specifies a type of object grouping and `Name` specifies the Mbean name.
domain_name	Optional. Name of the domain; for example, `mydomain`. If *domain_name* is not specified, the default domain name is used.

Example

```
java weblogic.Admin -url localhost:7001 -username adminuser
   -password gumby1234 CREATE -mbean
   "mydomain:Type=Server,Name=acctServer"
```

DELETE

Deletes a configuration Mbean. Returns `OK` in `stdout` when successful. This command cannot be used for run-time Mbeans.

Note: When you delete Mbeans, configuration objects are also deleted.

For more information about deleting Mbeans, see *Developing WebLogic Server Applications*, at `http://e-docs.bea.com/wls/docs60/programming/index.html`.

Syntax

```
java weblogic.Admin [–url URL] [-username username] [-password
   password] DELETE {–type mbean_type|–mbean mbean_name}
```

Arguments	Definition
mbean_type	Required. When deleting attributes for multiple objects of the same type.
mbean_name	Required. Fully qualified name of an Mbean, in the following format: "*domain*:Type=*type*,Name=*name*" `Type` specifies a type of object grouping, and `Name` specifies the Mbean name.

Example

```
java weblogic.Admin -url localhost:7001 -username adminuser
   -password gumby1234 DELETE -mbean
   "mydomain:Type:Server,Name=AcctServer"
```

GET

Displays run-time Mbean attributes. You can request a list of attributes for multiple objects of the same type by requesting attributes for the following:

- All Mbeans that belong to the same Mbean type:

  `GET {-pretty} -type mbean_type`

- A specific Mbean:

  `GET {-pretty} -mbean mbean_name`

The name of each of the specified Mbeans is included in the output. If `-pretty` is specified, each attribute name-value pair is displayed on a new line.

The `GET` command can only call run-time Mbeans.

The name-value pair for each attribute is specified within curly brackets. This format facilitates scripting by simplifying the parsing of the output.

The name of the Mbean is included in the output as follows:

```
{mbeanname mbean_name {property1 value} {property2 value}. . .}
{mbeanname mbean_name {property1 value} {property2 value} . . .}
. . .
```

If `-pretty` is specified, each attribute name-value pair is displayed on a new line. The name of each of the specified Mbeans is also included in the output, which is displayed as follows:

```
mbeanname: mbean_name
property1: value
property2: value
.
.
.
mbeanname: mbean_name
property1: value
property2: value
```

Syntax

```
java weblogic.Admin [–url URL] [-username username] [-password
   password] GET {-pretty} {–type mbean_type|–mbean mbean_name}
   [–property property1] [–property property2]...
```

Argument	Definition
mbean_type	Required. When getting attributes for multiple objects of the same type, output includes the name of the Mbean.
mbean_name	Fully qualified name of an Mbean, in the following format: *"domain*:Type=*type*,Location:*location*,Name=*name"* Type specifies a type of object grouping, Location specifies the location of the Mbean, and Name supplies the Mbean name.
pretty	Optional. Produces well-formatted output.
property	Optional. The name of the Mbean attribute or attributes to be listed. **Note:** If an attribute is not specified using this argument, all attributes are displayed.

Example

In the following example, a user requests a display of the Mbean attributes for a server named localhost, which is listening on port 7001:

```
java weblogic.Admin -url localhost:7001 GET -pretty -type Server
```

INVOKE

Invokes the specified method (including arguments) on the specified Mbean. This command can call only run-time Mbeans. Use this command to invoke methods that do not get or set Mbean attributes.

Syntax

```
java weblogic.Admin [–url URL] [-username username] [-password
   password] INVOKE {–type mbean_type|–mbean mbean_name} –method
   methodname [argument . . .]
```

Arguments	Definition
mbean_type	Required when invoking attributes for multiple objects of the same type, and must include the fully qualified name of the Mbean, as follows: "domain:Name:name,Type=*type*,Application=*application*"
mbean_name	Required. Fully qualified name of an Mbean, as follows: "domain:Type=*type*,Location=*location*,Name=*name*" where: ■ Type specifies the type of object grouping ■ Location specifies the location of the Mbean ■ Name is the Mbean name When the argument is a String array, the arguments must be passed in the following format: "*String1*;*String2*;. . . "
methodname	Required. Name of the method to be invoked. Following the method name, the user can specify arguments to be passed to the method call, as follows: "*domain*:Name=*name*,Type=*type*"

Example

The following example invokes an administration Mbean named admin_one using the method getAttributeStringValue:

```
java weblogic.Admin -username system -password gumby1234 INVOKE
   -mbean mydomain:Name=admin_one,Type=Administrator
   -method getAttributeStringValue PhoneNumber
```

SET

Sets the specified attribute values for the named configuration Mbean. Returns `OK` on `stdout` when successful. This command cannot be used for run-time Mbeans.

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 [–url URL] [-username username]
   [-password password] SET {–type mbean_type|–mbean mbean_name}
   –property property1 property1_value
   [-property property2 property2_value] . . .
```

Argument	Definition
mbean_type	Required when invoking properties for multiple objects of the same type, and must include the fully qualified name of the Mbean, as follows: "`domain:Name:name,Type=`*type*`,Application=`*application*"
mbean_name	Required. Must include the fully qualified name of an Mbean, in the following format: "*domain*`:Name=`*name*`,Location:`*location*`,Type=`*type*" where: ■ `Name` is the Mbean name ■ `Location` specifies the location of the Mbean ■ `Type` specifies the type of object grouping
property	Required. The name of the attribute property to be set.
property _value	Required. The value to be set with the attribute property. ■ When the argument is an Mbean array, the arguments must be passed in the following format: "*domain*`:Name=`*name*`,Type=`*type*`;`*domain*`:Name=`*name*`,Type=`*type*" ■ When the argument is a String array, the arguments must be passed in the following format: "*String1*`;`*String2*`;. . . "` ■ When setting the attribute properties for a JDBC Connection Pool, you must pass the arguments in the following format: "`user:`*username*`;password:`*password*`;server:`*servername*"

C Parameters for Web Server Plug-ins

This foloowing sections describe the parameters that you use to configure the Apache, Netscape, and Microsoft IIS Web server plug-ins:

Overview

You enter the parameters for each Web Server Plug-in special configuration files. Each Web Server has a different name for this configuration file and different rules for formatting the file. For details, see the following sections on each plug-in:

- "Installing and Configuring the Apache HTTP Server Plug-In" on page 9-1
- "Installing and Configuring the Microsoft Internet Information Server (ISAPI) Plug-In" on page 10-1
- "Installing and Configuring the Netscape Enterprise Server Plug-In (NSAPI)" on page 11-1

Enter Web server plug-ins parameters as described in the following table.

General Parameters for Web Server Plug-Ins

Note: Parameters are case sensitive.

Parameter	Default	Description
`WebLogicHost` (Required when proxying to a single WebLogic Server.)	`none`	WebLogic Server host (or virtual host name as defined in a Web Server running in WebLogic Server) to which HTTP requests should be forwarded. If you are using a WebLogic cluster, use the `WebLogicCluster` parameter instead of `WebLogicHost`.
`WebLogicPort` (Required when proxying to a single WebLogic Server.)	`none`	Port at which the WebLogic Server host is listening for WebLogic connection requests. (If you are using SSL between the plug-in and WebLogic Server, set this parameter to the SSL listen port (see "Configuring the Listen Port" on page 7-3) and set the `SecureProxy` parameter to `ON`). If you are using a WebLogic Cluster, use the `WebLogicCluster` parameter instead of `WebLogicPort`.

Parameter	Default	Description
`WebLogicCluster` (Required when proxying to a cluster of WebLogic Servers.)	`none`	List of WebLogic Servers that can be used in a cluster for load-balancing. The cluster list is a comma-delimited list of host:port entries. For example: `WebLogicCluster myweblogic.com:7001, yourweblogic.com:7001,theirweblogic.com:7001` If you are using SSL between the plug-in and WebLogic Server, set the port number to the SSL listen port (see "Configuring the Listen Port" on page 7-3) and set the `SecureProxy` parameter to `ON`. Use WebLogicCluster instead of the `WebLogicHost` and `WebLogicPort` parameters. WebLogic Server looks first for the `WebLogicCluster` parameter. If not found, it looks for and uses `WebLogicHost` and `WebLogicPort`. The plug-in does a simple round-robin between all available cluster members. The cluster list specified in this property is a starting point for the dynamic cluster list that the server and plug-in maintain. WebLogic Server and the plug-in work together to update the cluster list automatically with new, failed, and recovered cluster members. You can disable the use of the dynamic cluster list by setting the `DynamicServerList` parameter to `OFF` (Microsoft Internet Information Server only). The plug-in directs HTTP requests containing a cookie, URL-encoded session, or a session stored in the POST data to the server in the cluster that originally created the cookie.

`PathTrim`	`null`	String trimmed by the plug-in from the beginning of the original URL, before the request is forwarded to WebLogic Server. For example, if the URL
		`http://myWeb.server.com/weblogic/foo`
		is passed to the plug-in for parsing and if `PathTrim` has been set to strip off `/weblogic` before handing the URL to WebLogic Server, the URL forwarded to WebLogic Server is:
		`http://myweblogic.server.com:7001/foo`

`PathPrepend`	`null`	String that the plug-in prependeds to the beginning of the original URL, after `PathTrim` has been trimmed, and before the request is forwarded to WebLogic Server.

Parameter	Default	Description
ConnectTimeoutSecs	10	Maximum time in seconds that the plug-in should attempt to connect to the WebLogic Server host. Make the value greater than ConnectRetrySecs. If ConnectTimeoutSecs expires without a successful connection, even after the appropriate retries (see ConnectRetrySecs), an HTTP 503/Service Unavailable response is sent to the client. You can customize the error response by using the ErrorPage parameter.
ConnectRetrySecs	2	Interval in seconds that the plug-in should sleep between attempts to connect to the WebLogic Server host (or all of the servers in a cluster). Make this number less than the ConnectTimeoutSecs. The number of times the plug-in tries to connect before returning an HTTP 503/Service Unavailable response to the client is calculated by dividing ConnectTimeoutSecs by ConnectRetrySecs. To specify no retries, set ConnectRetrySecs equal to ConnectTimeoutSecs. However, the plug-in attempts to connect at least twice. You can customize the error response by using the ErrorPage parameter.

Parameter	Default	Description
`Debug`	`OFF`	Sets the type of logging performed for debugging operations. It is not advisable to switch on these debugging options in production systems. The debugging information is written to the `/tmp/wlproxy.log` file on UNIX systems and `c:\tmp\wlproxy.log` on Windows NT/2000 systems. You can set any of the following logging options (the `HFC`, `HTW`, `HFW`, and `HTC` options may be set in combination by entering them separated by commas, for example "`HFC,HTW`"): `ON` — The plug-in logs only informational and error messages. `OFF` — No debugging information is logged. `HFC` — The plug-in logs headers from the client, informational, and error messages. `HTW` — The plug-in logs headers sent to WebLogic Server, informational messages, and error messages. `HFW` — The plug-in logs headers sent from WebLogic Server, informational messages, and error messages. `HTC` — The plug-in logs headers sent to the client, informational messages, and error messages. `ALL` — The plug-in logs headers sent to and from the client, headers sent to and from WebLogic Server, information messages, and error messages.

Parameter	Default	Description
`DebugConfigInfo`	`OFF`	Enables the special query parameter "`__WebLogicBridgeConfig`". Use it to get details about configuration parameters from the plug-in. For example, if you enable "`__WebLogicBridgeConfig`" by setting `DebugConfigInfo` and then send a request that includes the query string `?__WebLogicBridgeConfig`, then the plug-in gathers the configuration information and run-time statistics and returns the information to the browser. The plug-in does not connect to the WebLogic Server in this case. This parameter is strictly for debugging and the format of the output message can change with releases. For security purposes, keep this parameter turned OFF in production systems.
`StatPath` (Not available for the Microsoft Internet Information Server Plug-In)	`false`	If set to `true`, the plug-in checks the existence and permissions of the translated path ("Proxy-Path-Translated") of the request before forwarding the request to WebLogic Server. If the file does not exist, an `HTTP 404 File Not Found` response is returned to the client. If the file exists but is not world-readable, an `HTTP 403/Forbidden` response is returned to the client. In either case, the default mechanism for the Web server to handle these responses fulfills the body of the response. This option is useful if both the WebLogic Server Web Application and the Web server have the same document root. You can customize the error response by using the `ErrorPage` parameter.
`ErrorPage`	`none`	You can create your own error page that is displayed when your Web server is unable to forward requests to WebLogic Server. You can set this parameter in one of two ways: ■ As a relative `URI` (file name). Depending on how you configure proxying (by MIME type or path) the request for the error page might be proxied to the WebLogic Server that is *not* responding. For this reason it is probablay more useful to specify an absolute URL. ■ As an absolute URL (recommended). Using an absolute URL to the error page will always proxy the request to the correct resource on your Web server or another WebLogic Server. For example: `http://host:port/myWebApp/ErrorPage.html`.

Parameter	Default	Description
`HungServerRecoverSecs`	`300`	Defines the amount of time the plug-in waits for a response to a request from WebLogic Server. The plug-in waits for `HungServerRecoverSecs` for the server to respond and then declares that server dead, and fails over to the next server. The value should be set to a very large value. If the value is less than the time the servlets take to process, then you may see unexpected results. Minimum value: `10`
`Idempotent`	`ON`	When set to `ON` and if the servers do not respond within `HungServerRecoverSecs`, the plug-ins fail over to the next server. For more information on fail over, see the section titled "Connection Errors and Clustering Failover" in the documentation for the plug-in If set to "`OFF`" the plug-ins do not fail over. If you are using the Netscape Enterprise Server Plug-In, or Apache HTTP Server you can set this parameter differently for different URLs or MIME types.
`CookieName`	`JSESSIONID`	If you change the name of the WebLogic Server session cookie in the WebLogic Server Web Application, you need to change the `CookieName` parameter in the plug-in to the same value. The name of the WebLogic session cookie is set in the WebLogic-specific deployment descriptor, in the <session-descriptor> (see http://e-docs.bea.com/wls/docs60/programming/weblogic_xml.html#session-descriptor) element.

Parameter	Default	Description
DefaultFileName	none	The plug-in performs the following steps: 1. Trims the path specified with the PathTrim parameter. 2. If the URI is "/" the plug-in appends the value of DefaultFileName. 3. Prepends the value specified with PathPrepend. This procedure prevents redirects from WebLogic Server. Set the DefaultFileName to the default welcome page of the Web Application in WebLogic Server to which requests are being proxied. For example, If the DefaultFileName is set to welcome.html, and PathTrim is set to /weblogic, an HTTP request like "http://somehost/weblogic" becomes "http://somehost/welcome.html". For this parameter to function, the same file must be specified as a welcome file in all the Web Applications to which requests are directed. For more information, see "Configuring Welcome Pages" in the *Administration Guide*. **Note for the Apache plug-in** using Stronghold or Raven: Do not define the DefaultFileName parameter in an IFmodule block. Instead, define it in a Location block.
MaxPostSize	-1	Maximum allowable size of POST data, in bytes. If the content-length exceeds MaxPostSize, the plug-in returns an error message. If set to -1, the size of POST data is not checked. This is useful for preventing denial-of-service attacks that attempt to overload the server with POST data.

Parameter	Default	Description

`MatchExpression`
(Apache HTTP Server only)
	`none`	When proxying by MIME type, set the filename pattern inside of an `IfModule` block using the `MatchExpression` parameter.
		Example when proxying by MIME type:
		`<IfModule mod_weblogic.c>`
		`  MatchExpression *.jsp`
		`  WebLogicHost=myHost|paramName=value`
		`</IfModule>`
		Example when proxying by path:
		`<IfModule mod_weblogic.c>`
		`  MatchExpression /weblogic`
		`  WebLogicHost=myHost|paramName=value`
		`</IfModule>`

`FileCaching`	`ON`	When set to `ON`, and the size of the POST data in a request is greater than 2048 bytes, the POST data is stored on disk in a temporary file and forwarded to WebLogic Server in chunks of 8192 bytes. Setting `FileCaching` to `ON`, however, can cause a problem with the progress bar displayed by a browser that indicates the progress of a download. The browser shows that the download has completed even though the file is still being transferred. When set to `OFF` and size of the POST data in a request is greater than 2048 bytes, the POST data is stored in memory and sent to WebLogic Server in chunks of 8192 bytes. Setting to `OFF` causes problems if the server goes down while processing the request because the plug-in is not able to fail over.
`WlForwardPath` (Microsoft Internet Information Server only)	null	If `WlForwardPath` is set to "/" all requests are proxied. To forward any requests starting with a particular string, set `WlForwardPath` to the string. For example, setting `WlForwardPath` to `/weblogic` forwards all requests starting with `/weblogic` to Weblogic Server. This parameter is required if you are proxying by path. You can set multiple strings by separating the strings with commas. For example: `WlForwardPath=/weblogic,/bea`.

Parameter	Default	Description
`MaxSkips` (Microsoft Internet Information Server only)	10	Valid only if `DynamicServerList` is set to `OFF`. If a WebLogic Server listed in either the `WebLogicCluster` parameter or a dynamic cluster list returned from WebLogic Server fails, the failed server is marked as "bad" and the plug-in attempts to connect to the next server in the list. `MaxSkips` sets the number of attempts after which the plug-in will retry the server marked as "bad".
`DynamicServerList` (Microsoft Internet Information Server only	ON	When set to `OFF`, the plug-in ignores the dynamic cluster list used for load balancing requests proxied from the plug-in and only uses the static list specified with the `WebLogicCluster` parameter. Normally this parameter should remain set to `ON`. There are some implications for setting this parameter to `OFF`: ■ If one or more servers in the static list fails, the plug-in could waste time trying to connect to a dead server, resulting in decreased performance. ■ If you add a new server to the cluster, the plug-in can not proxy requests to the new server unless you re-define this parameter. WebLogic Server automatically adds new servers to the dynamic server list when they become part of the cluster.

SSL Parameters for Web Server Plug-Ins

Note: Parameters are case sensitive.

Parameter	Default	Description
SecureProxy	OFF	Set this parameter to ON to enable the use of the SSL protocol for all communication between the WebLogic Server proxy plug-in and WebLogic Server. Remember to configure a port on the corresponding WebLogic Server for the SSL protocol before defining this parameter. This parameter may be set at two levels,: in the configuration for the main server and, if you have defined any virtual hosts, in the configuration for the virtual host. The configuration for the virtual host inherits the SSL configuration from the configuration of the main server if the setting is not overridden in the configuration for the virtual host.
TrustedCAFile	none	Name of the file that contains the digital certificates for the trusted certificate authorities for the WebLogic Server proxy plug-in. This parameter is required if the SecureProxy parameter is set to ON. The filename must include the full directory path of the file.
RequireSSLHostMatch	true	Determines whether the host name to which the WebLogic Server proxy plug-in is connecting must match the Subject Distinguished Name field in the digital certificate of the WebLogic Server to which the proxy plug-in is connecting.
SSLHostMatchOID	22	The ASN.1 Object ID (OID) that identifies which field in the Subject Distinguished Name of the peer digital certificate is to be used to perform the host match comparison. The default for this parameter corresponds to the CommonName field of the Subject Distinguished Name. Common OID values are: ■ Sur Name—23 ■ Common Name—22 ■ Email—13 ■ Organizational Unit—30 ■ Organization—29 ■ Locality—26

Index

D

K

L

M

N

O

P

R

S

T

U

V

W