BEA WebLogic Server™

WebLogic Server Command Reference

Version 8.1
Revised: June 28, 2006

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Contents

About This Document

Audience xv

e-docs Web Site xvi

How to Print the Document xvi

Related Information xvi

Contact Us! xvi

Documentation Conventions xvii

1. weblogic.Admin Command-Line Reference

Required Environment for the weblogic.Admin Utility 1-2

Syntax for Invoking the weblogic.Admin Utility 1-3

SSL Arguments 1-3

Using SSL to Secure Administration Requests: Main Steps 1-5

Specifying Trust for weblogic.Admin 1-6

Specifying Host Name Verification for weblogic.Admin 1-6

Connection Arguments 1-7

User Credentials Arguments 1-10

Specifying User Credentials 1-12

Examples of Providing User Credentials. 1-13

Protocol Support 1-13

Example Environment 1-14

Exit Codes Returned by weblogic.Admin 1-14

Command for Storing User Credentials 1-15

STOREUSERCONFIG. 1-15

Syntax 1-15

Configuring the Default Path Name 1-17

Creating User-Configuration and Key Files 1-17

Using a Single Key File for Multiple User-Configuration Files. 1-18

Examples. 1-19

Commands for Managing the Server Life Cycle 1-19

CANCEL_SHUTDOWN 1-21

Syntax 1-21

Example 1-21

DISCOVERMANAGEDSERVER 1-21

Syntax 1-22

Example 1-23

FORCESHUTDOWN. 1-24

Syntax 1-24

Example 1-27

LOCK 1-28

Syntax 1-28

Example 1-28

RESUME 1-29

Syntax 1-29

Example 1-29

SHUTDOWN 1-30

Syntax 1-30

Example 1-34

START 1-35

Syntax 1-36

Example. 1-36

STARTINSTANDBY 1-37

Syntax 1-37

Example. 1-38

UNLOCK 1-39

Syntax 1-39

Example. 1-39

Commands for Retrieving Information about WebLogic Server and Server Instances . 1-39

CONNECT 1-40

Syntax 1-41

Example. 1-41

GETSTATE 1-42

Syntax 1-42

Example. 1-43

HELP. 1-43

Syntax 1-43

Example. 1-43

LICENSES 1-44

Syntax 1-44

Example. 1-44

LIST 1-44

Syntax 1-45

Example. 1-45

PING 1-46

Syntax 1-46

Example. 1-46

SERVERLOG 1-47

Syntax 1-47

Example 1-48

THREAD_DUMP. 1-49

Syntax 1-49

Example 1-50

VERSION 1-50

Syntax 1-50

Example 1-50

Commands for Managing JDBC Connection Pools 1-52

CREATE_POOL. 1-53

Syntax 1-53

Example 1-55

DESTROY_POOL 1-56

Syntax 1-56

Example 1-57

DISABLE_POOL 1-57

Syntax 1-57

Example 1-58

ENABLE_POOL 1-58

Syntax 1-59

Example 1-59

TEST_POOL 1-59

Syntax 1-60

Example 1-60

RESET_POOL 1-60

Syntax 1-61

Example 1-61

EXISTS_POOL 1-61

Syntax 1-61

Example. 1-62

Commands for Managing WebLogic Server MBeans. 1-62

Specifying MBean Types 1-62

MBean Management Commands 1-63

CREATE 1-64

Syntax 1-64

Example. 1-65

DELETE 1-66

Syntax 1-66

Example. 1-67

GET 1-68

Syntax 1-69

Example. 1-70

INVOKE 1-70

Syntax 1-71

Example. 1-72

QUERY 1-72

Syntax 1-73

Example. 1-74

SET 1-75

Syntax 1-75

Example. 1-77

Running Commands in Batch Mode 1-78

BATCHUPDATE 1-78

Syntax 1-79

Example. 1-80

Commands for Working with Clusters 1-80

CLUSTERSTATE 1-81

Syntax 1-81

Example 1-82

MIGRATE. 1-82

Syntax 1-83

Examples. 1-84

STARTCLUSTER. 1-84

Syntax 1-85

Example 1-85

STOPCLUSTER 1-86

Syntax 1-86

Example 1-86

VALIDATECLUSTERCONFIG. 1-87

Syntax 1-87

Example 1-88

Command for Purging Tasks. 1-88

PURGETASKS 1-88

Syntax 1-88

Examples. 1-89

2. Using Command-Line Utilities to Configure a WebLogic Server Domain

Using Command-Line Utilities to Clone the MedRec Domain: Main Steps 2-2

Setting Up the Environment. 2-3

Creating a Domain 2-3

Creating JDBC Resources 2-4

Using the Sample Commands. 2-5

Creating JMS Resources 2-6

Creating Mail Resources 2-9

Creating and Configuring Managed Servers 2-9
Configuring Machines and Node Manager Properties 2-10
Deploying Applications 2-12
Sample Korn Shell Script 2-13
Using weblogic.Admin Commands to Manage Users and Groups 2-20
Finding the Object Name for an AuthenticationProvider MBean 2-21
Creating a User 2-22
Adding a User to a Group. 2-22
Verifying Whether a User is a Member of a Group 2-22
Listing Groups to Which a User Belongs. 2-23
Limiting Group Membership Searching in an LDAP Server. 2-23
Listing Users and Groups in a Security Realm 2-24
Example: Korn Shell Script for Listing Users 2-25
Changing a Password 2-27
Protecting User Accounts in a Security Realm 2-27
Using weblogic.Admin Commands to Target Resources 2-29
Using weblogic.Admin Commands to Configure the WebLogic SNMP Agent. 2-35
Using the Sample Commands 2-37
Using weblogic.Admin Commands to Create a Simple Cluster 2-38
Deploying Applications and Starting Servers in the Simple Cluster 2-41

3. Using the WebLogic Server Java Utilities

appc 3-3
AppletArchiver 3-3
Syntax 3-3
autotype 3-4
BuildXMLGen 3-4
CertGen 3-4

Syntax 3-4

Example 3-6

ClientDeployer 3-6

clientgen 3-7

Conversion 3-7

der2pem 3-7

Syntax 3-7

Example 3-8

dbping 3-8

Creating a DB2 Package with dbping. 3-9

Syntax 3-9

Example 3-11

ddcreate. 3-12

DDInit 3-12

Limitations 3-13

Example 3-13

Deployer 3-15

ejbc 3-15

EJBGen 3-15

encrypt 3-15

Syntax 3-16

Examples. 3-16

getProperty 3-17

Syntax 3-17

Example 3-17

host2ior 3-18

Syntax 3-18

ImportPrivateKey 3-18

Syntax 3-18

Example. 3-19

jhtml2jsp 3-20

Syntax 3-21

jspc 3-21

logToZip 3-21

Syntax 3-21

Examples 3-22

MBean Commands 3-22

MulticastTest 3-22

Syntax 3-23

Example. 3-23

myip. 3-24

Syntax 3-24

Example. 3-24

pem2der 3-25

Syntax 3-25

Example. 3-25

Pointbase and the Pointbase ant Task 3-25

rmic 3-26

Schema 3-26

Syntax 3-26

Example. 3-26

servicegen 3-27

SearchAndBuild 3-27

Example. 3-27

showLicenses 3-28

Syntax 3-28

Example 3-28
source2wsdd 3-28
system 3-28
Syntax 3-28
Example 3-28
ValidateCertChain 3-29
verboseToZip 3-30
Syntax 3-30
Example 3-30
wlappc 3-30
wlcompile 3-31
wlconfig 3-31
wldeploy 3-31
wlpackage 3-31
wlserver. 3-31
writeLicense 3-31
Syntax 3-32
Examples. 3-32
wsdl2Service 3-33
wsdlgen 3-34
wspackage 3-34

4. weblogic.Server Command-Line Reference

Required Environment and Syntax for weblogic.Server. 4-1
Environment. 4-2
Modifying the Classpath 4-2
Syntax. 4-3
Default Behavior 4-3

weblogic.Server Configuration Options 4-4
JVM Parameters 4-5
Location of License and Configuration Data 4-6
Examples 4-8
Server Communication 4-9
SSL 4-14
Setting Additional SSL Attributes. 4-17
Security. 4-17
Message Output and Logging. 4-22
Setting Logging Attributes 4-23
Other Server Configuration Options 4-24
Clusters. 4-27
Using the weblogic.Server Command Line to Start a Server Instance 4-27
Using the weblogic.Server Command Line to Create a Domain. 4-28
Verifying Attribute Values That Are Set on the Command Line 4-29

5. Using Ant Tasks to Configure a WebLogic Server Domain

Overview of Configuring and Starting Domains Using Ant Tasks. 5-1
Starting Servers and Creating Domains Using the wlserver Ant Task 5-2
What the wlserver Ant Task Does 5-2
Basic Steps for Using wlserver. 5-2
Sample build.xml Files for wlserver 5-3
wlserver Ant Task Reference 5-4
Configuring a WebLogic Server Domain Using the wlconfig Ant Task 5-6
What the wlconfig Ant Task Does 5-6
Basic Steps for Using wlconfig 5-7
Sample build.xml Files for wlconfig 5-7
Complete Example 5-7

Query and Delete Example 5-10
Example of Setting Multiple Attribute Values 5-10
wlconfig Ant Task Reference 5-11
Main Attributes 5-11
Nested Elements 5-13

6. WebLogic SNMP Agent Command-Line Reference

Required Environment and Syntax for the SNMP Command-Line Interface 6-2
Environment 6-2
Common Arguments 6-2
Commands for Retrieving the Value of WebLogic Server Attributes 6-4
snmpwalk 6-4
Syntax 6-5
Example 6-5
snmpgetnext 6-6
Syntax 6-6
Example 6-7
snmpget 6-8
Syntax 6-8
Example 6-9
Commands for Testing Traps 6-9
snmpv1trap 6-10
Syntax 6-10
Example 6-12
snmptrapd 6-13
Syntax 6-13
Example 6-13
Example: Sending Traps to the Trap Daemon 6-13

About This Document

This document [[introduces BEA WebLogic Server™ features and describes the architecture of applications that run on the WebLogic Server platform.]]

The document is organized as follows:

- Chapter 1, "weblogic.Admin Command-Line Reference,"describes using the `weblogic.Admin` command to configure a WebLogic Server domain from a command shell or a script.
- Chapter 2, "Using Command-Line Utilities to Configure a WebLogic Server Domain,"
- Chapter 3, "Using the WebLogic Server Java Utilities," describes various Java utilities you can use to manage and troubleshoot a WebLogic Server domain.
- Chapter 4, "weblogic.Server Command-Line Reference," describes how to start WebLogic Server instances from a command shell or from a script.
- Chapter 5, "Using Ant Tasks to Configure a WebLogic Server Domain,"
- Chapter 6, "WebLogic SNMP Agent Command-Line Reference,"

Audience

This document is written for system administrators and application developers deploying e-commerce applications using the Java 2 Platform, Enterprise Edition (J2EE) from Sun Microsystems. It is assumed that readers are familiar with Web technologies and the operating system and platform where WebLogic Server is installed.

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.

Related Information

- Creating and configuring WebLogic Servers and Domains
- Managing a WebLogic Server Domain
- Administration Console Online Help

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 the user is told to enter from the keyboard.
	Examples:
	`import java.util.Enumeration;`
	`chmod u+w *`
	`config/examples/applications`
	`.java`
	`config.xml`
	`float`

`monospace italic text`	Placeholders. *Example*: `String` *`CustomerName`*`;`

`UPPERCASE MONOSPACE TEXT`	Device names, environment variables, and logical operators.
	*Example*s:
	`LPT1`
	`BEA_HOME`
	`OR`

Convention	Usage
`{ }`	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.

weblogic.Admin Command-Line Reference

The weblogic.Admin utility is a command-line interface that you can use to administer, configure, and monitor WebLogic Server.

Like the Administration Console, for most commands this utility assumes the role of client that invokes administrative operations on the Administration Server, which is the central management point for all servers in a domain. (All Managed Servers retrieve configuration data from the Administration Server, and the Administration Server can access runtime data from all Managed Servers.) While the Administration Console interacts only with the Administration Server, the weblogic.Admin utility can access the Administration Server as well as all active server instances directly. If the Administration Server is down, you can still use the weblogic.Admin utility to retrieve runtime information from Managed Servers and invoke some administrative commands. However, you can save configuration changes to the domain's config.xml file only when you access the Administration Server.

To automate administrative tasks, you can invoke the weblogic.Admin utility from shell scripts. If you plan to invoke this utility multiple times from a shell script, consider using the BATCHUPDATE command, which is described in "Running Commands in Batch Mode" on page 1-78.

The following sections describe using the weblogic.Admin utility:

- "Required Environment for the weblogic.Admin Utility" on page 1-2
- "Syntax for Invoking the weblogic.Admin Utility" on page 1-3
- "Command for Storing User Credentials" on page 1-15

- "Commands for Managing the Server Life Cycle" on page 1-19
- "Commands for Retrieving Information about WebLogic Server and Server Instances" on page 1-39
- "Commands for Managing JDBC Connection Pools" on page 1-52
- "Commands for Managing WebLogic Server MBeans" on page 1-62
- "Running Commands in Batch Mode" on page 1-78
- "Commands for Working with Clusters" on page 1-80
- "Command for Purging Tasks" on page 1-88

For more information, see:

- "Deployment Tools Reference" in *Deploying WebLogic Server Applications*. Describes how to deploy J2EE modules on a WebLogic Server instance using the `weblogic.Deployer` command-line utility.
- "Using Command-Line Utilities to Configure a WebLogic Server Domain" on page 2-1. Provides extended examples of using `weblogic.Admin` commands to configure a WebLogic Server domain.

Required Environment for the weblogic.Admin Utility

To set up your environment for the `weblogic.Admin` utility:

1. Install and configure the WebLogic Server software, as described in the WebLogic Server Installation Guide.

2. Add WebLogic Server classes to the `CLASSPATH` environment variable and `WL_HOME\server\bin` to the `PATH` environment variable.

   You can use a `WL_HOME\server\bin\setWLSenv` script to set both variables. See "Modifying the Classpath" on page 4-2.

3. If you want the `weblogic.Admin` utility to use a listen port that is reserved for administration traffic, you must configure a domain-wide administration port as described in "Enabling the Domain-Wide Administration Port" in the Administration Console Online Help.

   The domain-wide administration port is secured by SSL. For information about using secured ports with the `weblogic.Admin` utility, refer to "SSL Arguments" on page 1-3.

Note: If a server instance is deadlocked, it can respond to `weblogic.Admin` commands only if you have enabled the domain-wide administration port. If you have not already enabled the domain-wide administration port, your only option is to shut down the server instance by killing the Java process that is running the server. You will lose all session data. For information on enabling the domain-wide administration port, refer to "Enabling the Domain-Wide Administration Port" in the Administration Console Online Help.

Syntax for Invoking the weblogic.Admin Utility

```
java [ SSL Arguments ]
     weblogic.Admin
     [ Connection Arguments ]
     [ User Credentials Arguments ]
     COMMAND-NAME command-arguments
```

The command names and arguments are not case sensitive.

The following sections provide detailed syntax information:

- "SSL Arguments" on page 1-3
- "Connection Arguments" on page 1-7
- "User Credentials Arguments" on page 1-10
- "Protocol Support" on page 1-13

Note: Both the weblogic.Deployer tool and the BEA WebLogic Scripting Tool (WLST) also use the SSL arguments, Connection arguments, and User Credentials arguments.

SSL Arguments

```
java [ -Dweblogic.security.TrustKeyStore=DemoTrust ]
     [ -Dweblogic.security.JavaStandardTrustKeystorePassPhrase=password ]
     [ -Dweblogic.security.CustomTrustKeyStoreFileName=filename
       -Dweblogic.security.TrustKeystoreType=CustomTrust
       [-Dweblogic.security.CustomTrustKeystorePassPhrase=password ]
     ]
     [ -Dweblogic.security.SSL.hostnameVerifier=classname ]
     [ -Dweblogic.security.SSL.ignoreHostnameVerification=true ]
     weblogic.Admin
     [ User Credentials Arguments ]
     COMMAND-NAME command-arguments
```

If you have enabled the domain-wide administration port, or if you want to secure your administrative request by using some other listen port that is secured by SSL, you must include SSL arguments when you invoke `weblogic.Admin`. Table 1-1 describes all SSL arguments for the `weblogic.Admin` utility.

Table 1-1 SSL Arguments

Argument	Definition
`-Dweblogic.security.TrustKeyStore=DemoTrust`	Causes `weblogic.Admin` to trust the CA certificates in the demonstration trust keystore (`WL_HOME\server\lib\DemoTrust.jks`). This argument is required if the server instance to which you want to connect is using the demonstration identity and certificates. By default, `weblogic.Admin` trusts only the CA certificates in the Java Standard Trust keystore (`SDK_HOME\jre\lib\security\cacerts`).
`-Dweblogic.security.JavaStandardTrustKeystorePassPhrase=`*`password`*	Specifies the password that was used to secure the Java Standard Trust keystore. If the Java Standard Trust keystore is protected by a password, and if you want to trust its CA certificates, you must use this argument. By default, the Java Standard Trust keystore is not protected by a password.

`-Dweblogic.security.CustomTrustKeyStoreFileName=`*`filename`*
`-Dweblogic.security.TrustKeystoreType=CustomTrust`
	Causes `weblogic.Admin` to trust the CA certificates in a custom keystore that is located at *`filename`*. You must use both arguments to trust custom keystores.

`-Dweblogic.security.CustomTrustKeystorePassPhrase=`*`password`*	Specifies the password that was used to secure the custom keystore. You must use this argument only if the custom keystore is protected by a password.

Table 1-1 SSL Arguments

Argument	Definition
`-Dweblogic.security.SSL.hostnameVerifier=`*`classname`*	Specifies the name of a custom Host Name Verifier class. The class must implement the `weblogic.security.SSL.HostnameVerifier` interface.
`-Dweblogic.security.SSL.ignoreHostnameVerification=true`	Disables host name verification.

Using SSL to Secure Administration Requests: Main Steps

To secure administration requests with SSL:

1. Ensure that two-way SSL is disabled on the server instance to which you want to connect.

   See "Using the SSL Protocol to Connect to WebLogic Server from weblogic.Admin" in *Managing WebLogic Security.*

   By default, when you enable SSL, a server instance supports one-way SSL. Because two-way SSL provides additional security, you might have enabled two-way SSL. However, `weblogic.Admin` does not support two-way SSL.

2. Ensure that the trusted CA certificates are stored in a keystore that the `weblogic.Admin` utility can access through the file system.

3. When you invoke the `weblogic.Admin` utility, include arguments that specify the following:

   – A secure protocol and port.

     See “Protocol Support” on page 1-13.

   – (Optional) The trusted CA certificates and certificate authorities.

     See “Specifying Trust for weblogic.Admin” on page 1-6.

   – (Optional) A host name verifier.

     See “Specifying Host Name Verification for weblogic.Admin” on page 1-6.

Specifying Trust for weblogic.Admin

When the `weblogic.Admin` utility connects to a server's SSL port, it must specify a set of certificates that describe the certificate authorities (CAs) that the utility trusts. See "Private Keys, Digital Certificates and Trusted Certificate Authorities" in *Managing WebLogic Security*.

To specify trust for `weblogic.Admin`:

- To trust only the CA certificates in the Java Standard Trust keystore, you do not need to specify command-line arguments, unless the keystore is protected by a password.

  If the Java Standard Trust keystore is protected by a password, use the following command-line argument:

  `-Dweblogic.security.JavaStandardTrustKeystorePassPhrase=`*`password`*

- To trust both the CA certificates in the Java Standard Trust keystore **and** in the demonstration trust keystore, include the following argument:

  `-Dweblogic.security.TrustKeyStore=DemoTrust`

  This argument is required if the server instance to which you want to connect is using the demonstration identity and certificates.

  If the Java Standard Trust keystore is protected by a password, include the following command-line argument:

  `-Dweblogic.security.JavaStandardTrustKeystorePassPhrase=`*`password`*

- To trust only the CA certificates in a keystore that you create, specify the following command-line arguments:

  - `-Dweblogic.security.CustomTrustKeyStoreFileName=`*`filename`*

    where *`filename`* specifies the fully qualified path to the trust keystore.

  - `-Dweblogic.security.TrustKeystoreType=CustomTrust`

    This optional command-line argument specifies the type of the keystore. Generally, this value for type is `jks`.

  - If the custom keystore is protected by a password, include `-Dweblogic.security.CustomTrustKeystorePassPhrase=`*`password`*

Specifying Host Name Verification for weblogic.Admin

A host name verifier ensures the host name URL to which the client connects matches the host name in the digital certificate that the server sends back as part of the SSL connection. A host name verifier is useful when an SSL client, or a SSL server acting as a client, connects to an application server on a remote host. It helps to prevent man-in-the-middle attacks. See "Using Host Name Verification" in *Managing WebLogic Security*.

To specify host name verification for `weblogic.Admin`:

- To use the host name verifier that the WebLogic Security Service provides, you do not need to specify host-name verification arguments.

  Note: If you specify an IP address or the `localhost` string in the `weblogic.Admin -url` or `-adminurl` argument, the host name verifier that the WebLogic Security Service provides will allow the connection if the CN field of the digital certificate matches the DNS name of the local host.

- To use a custom host name verifier, specify:
  `-Dweblogic.security.SSL.hostnameVerifier=`*`classname`*

  where *`classname`* specifies the implementation of the `weblogic.security.SSL.HostnameVerifier` interface.

- To disable host name verification, specify:
  `-Dweblogic.security.SSL.ignoreHostnameVerification=true`

Connection Arguments

```
java [ SSL Arguments ]
     weblogic.Admin
     [ {-url URL} | {-adminurl URL} ]
     [ User Credentials Arguments ]
     COMMAND-NAME command-arguments
```

When you invoke most `weblogic.Admin` commands, you specify the arguments in Table 1-2 to connect to a WebLogic Server instance. Some commands have special requirements for the connection arguments. Any special requirements are described in the command documentation.

Table 1-2 Connection Arguments

Argument	Definition
`-url [protocol://]listen-address:listen-port`	The listen address and listen port of the server instance that runs the command. In most cases, you should specify the Administration Server's address and port, which is the central management point for all servers in a domain. Some commands, such as START and CREATE, **must** run on the Administration Server. The documentation for each command indicates whether this is so. If you specify a Managed Server's listen address and port, the command can access data only for that server instance; you cannot run a command on one Managed Server to view or change data for another server instance. When you use MBean-related commands, you must specify the Administration Server's listen address and port to access Administration MBeans. To access Local Configuration MBeans or Runtime MBeans, you can specify the server instance on which the MBeans reside. (However, the `-adminurl` argument can also retrieve Local Configuration MBeans or Runtime MBeans from any server.) For more information on where MBeans reside, refer to "WebLogic Server Managed Resources and MBeans" in the *Programming WebLogic Management Services with JMX* guide. To use a listen port that is not secured by SSL, the format is `-url [protocol://]listen-address:port` To use a port that is secured by SSL, the format is `-url secure-protocol://listen-address:port` If you have set up a domain-wide administration port, you must specify the administration port number: `-url secure-protocol://listen-address:domain-wide-admin-port` For information about valid values for `protocol` and `secure-protocol`, refer to "Protocol Support" on page 1-13. For more information about the listen address and listen ports, refer to "`-Dweblogic.ListenAddress=host`" on page 4-12 and "`-Dweblogic.ListenPort= portnumber`" on page 4-12. For more information about the domain-wide administration port, refer to "Enabling the Domain-Wide Administration Port" in the Administration Console Online Help. The default value for this argument is `t3://localhost:7001`.

Table 1-2 Connection Arguments (Continued)

Argument	Definition
`-adminurl [protocol://]Admin-Server-listen-address:listen-port`	Enables the Administration Server to retrieve Local Configuration MBeans or Runtime MBeans for any server instance in the domain. For information about types of MBeans, refer to "WebLogic Server Managed Resources and MBeans" in the *Programming WebLogic Management Services with JMX* guide. For all commands other than the MBean commands, `-adminurl` *`admin-address`* and `-url` *`admin-address`* are synonymous. The `-adminurl` value must specify the listen address and listen port of the Administration Server. To use a port that is not secured by SSL, the format is `-adminurl [`*`protocol`*`]`*`Admin-Server-listen-address`*`:`*`port`*. To use a port that is secured by SSL, the format is `-adminurl` *`secure-protocol`*`://`*`Admin-Server-listen-address`*`:`*`port`* If you have set up a domain-wide administration port, you must specify the administration port number: `-adminurl` *`secure-protocol`*`://`*`Admin-Server-listen-address`*`:`*`domain-wide-admin-port`* For information about valid values for *`protocol`* and *`secure-protocol`*, refer to "Protocol Support" on page 1-13. There is no default value for this argument.

User Credentials Arguments

```
java [ SSL Arguments ]
     weblogic.Admin
     [ Connection Arguments ]
     [ { -username username [-password password] } |
       [ -userconfigfile config-file [-userkeyfile admin-key] ]
     ]
     COMMAND-NAME command-arguments
```

When you invoke most `weblogic.Admin` commands, you specify the arguments in Table 1-3 to provide the user credentials of a WebLogic Server user who has permission to invoke the command.

Table 1-3 User Credentials Arguments

Argument	Definition
`-username` *`username`*	The name of the user who is issuing the command. This user must have appropriate permission to view or modify the target of the command. For information about permissions for system administration tasks, refer to "Security Roles" in the *Securing WebLogic Resources* guide.
`-password` *`password`*	The password that is associated with the username. If you do not specify the `-password` argument, `weblogic.Admin` prompts you for a password. If *`WL_HOME`*`\server\bin` is specified in the `PATH` environment variable, `weblogic.Admin` uses a set of WebLogic Server libraries that prevent the password from being echoed to standard out. For information on setting environment variables, see "Required Environment for the weblogic.Admin Utility" on page 1-2.
`-userconfigfile` *`config-file`*	Specifies the name and location of a user-configuration file, which contains an encrypted username and password. The encrypted username must have permission to invoke the command you specify. If you do not specify `-userconfigfile` *`config-file`*, and if you do not specify `-username` *`username`*, `weblogic.Admin` searches for a user-configuration file at the default path name. (See "STOREUSERCONFIG" on page 1-15.)
`-userkeyfile` *`admin-key`*	Specifies the name and location of the key file that is associated with the user-configuration file you specify. When you create a user-configuration file, the `STOREUSERCONFIG` command uses a key file to encrypt the username and password. Only the key file that encrypts a user-configuration file can decrypt the username and password. If you do not specify `-userkeyfile` *`admin-key`*, `weblogic.Admin` searches for a key file at the default path name. (See "STOREUSERCONFIG" on page 1-15.)

Note: The exit code for all commands is 1 if the Administration client cannot connect to the server or if the WebLogic Server instance rejects the username and password.

Specifying User Credentials

The simplest way to specify user credentials is to create a user configuration file and key file in the default location. Thereafter, you do not need to include user credentials in `weblogic.Admin` invocations. A user-configuration file contains encrypted user credentials that can be decrypted only by a single key file. See "STOREUSERCONFIG" on page 1-15.

For example, the following command creates a user configuration file and key file in the default location:

```
java weblogic.Admin -username weblogic -password weblogic STOREUSERCONFIG
```

After you enter this `STOREUSERCONFIG` command, you can invoke `weblogic.Admin` without specifying credentials on the command line or in scripts. For example:

```
java weblogic.Admin GET -pretty -type -Domain
```

If you create a user configuration file or key file in a location other than the default, you can include the `-userconfigfile` *`config-file`* and `-userkeyfile` *`admin-key`* arguments on the command line or in scripts.

If you do not create a user configuration file and key file, you must use the `-username` and `-password` arguments when invoking the `weblogic.Admin` utility directly on the command line or in scripts. With these arguments, the username and password are not encrypted. If you store the values in a script, the user credentials can be used by anyone who has read access to the script.

The following list summarizes the order of precedence for the `weblogic.Admin` user-credentials arguments:

- If you specify `-username` *`username`* `-password` *`password`*, the utility passes the unencrypted values to the server instance you specify in the `-url` argument.

  These arguments take precedence over the `{ -userconfigfile` *`config-file`* `-userkeyfile` *`admin-key`* `}` arguments.

- If you specify `-username` *`username`*, the utility prompts for a password. Then it passes the unencrypted values to the server instance you specify in the `-url` argument.

  This argument also takes precedence over the `{ -userconfigfile` *`config-file`* `-userkeyfile` *`admin-key`* `}` arguments.

- If you specify `{ -userconfigfile` *`config-file`* `-userkeyfile` *`admin-key`* `}` and do not specify `{ -username` *`username`* `[-password` *`password`*`]}`, the utility passes the values that are encrypted in *`config-file`* to the server instance you specify in the `-url` argument.

- If you specify neither `{ -username username [-password password] }` nor `{ -userconfigfile config-file -userkeyfile admin-key }`, the utility searches for a user-configuration file and key file at the default path names. The default path names vary depending on the JVM and the operating system. See "Configuring the Default Path Name" on page 1-17.

Examples of Providing User Credentials

The following command specifies the username **weblogic** and password **weblogic** directly on the command line:

```
java weblogic.Admin -username weblogic -password weblogic COMMAND
```

The following command uses a user-configuration file and key file that are located at the default pathname:

```
java weblogic.Admin COMMAND
```

See "Configuring the Default Path Name" on page 1-17.

The following command uses a user-configuration file named `c:\wlUser1-WebLogicConfig.properties` and a key file named `e:\secure\myKey`:

```
java -userconfigfile c:\wlUser1-WebLogicConfig.properties
-userkeyfile e:\secure\myKey COMMAND
```

Protocol Support

The `-url` and `-adminurl` arguments of the `weblogic.Admin` utility support the `t3`, `t3s`, `http`, `https`, and `iiop` protocols.

If you want to use `http` or `https` to connect to a server instance, you must enable HTTP Tunneling for that instance. For more information, refer to "Configuring the HTTP Protocol" in the Administration Console Online Help.

If you want to use `iiop` to connect to a server instance, you must enable the `iiop` protocol for that instance. For more information, refer to "Enabling and Configuring the IIOP Protocol" in the Administration Console Online Help.

If you use the `-url` argument to specify a non-secured port, the `weblogic.Admin` utility uses `t3` by default. For example, `java weblogic.Admin -url localhost:7001` resolves to `java weblogic.Admin -url` **`t3://`**`localhost:7001`.

If you use either the `-url` or `-adminurl` argument to specify a port that is secured by SSL, you must specify either `t3s` or `https`. See "Using SSL to Secure Administration Requests: Main Steps" on page 1-5.

Example Environment

In many of the examples throughout the sections that follow, it is assumed that a certain environment has been set up:

- The WebLogic Server administration domain is named MedRec.
- The Administration Server is named MedRecServer and listens on port 7001.
- The Administration Server uses the name of its host machine, AdminHost, as its listen address. For more information about the listen address and listen ports, refer to "`-Dweblogic.ListenAddress=host`" on page 4-12 and "`-Dweblogic.ListenPort=portnumber`" on page 4-12.
- The `weblogic` username has system-administrator privileges and uses `weblogic` for a password.
- The user credentials have **not** been encrypted in a user configuration file.
- A Managed Server named MedRecManagedServer uses the name of its host machine, ManagedHost, as its listen address and 8001 as its listen port.

Exit Codes Returned by weblogic.Admin

All `weblogic.Admin` commands return an exit code of `0` if the command succeeds and an exit code of `1` if the command fails.

To view the exit code from a Windows command prompt, enter `echo %ERRORLEVEL%` after you run a `weblogic.Admin` command. To view the exit code in a `bash` shell, enter `echo $?`.

For example:

```
D:\>java weblogic.Admin -username weblogic -password weblogic GET -pretty
-mbean "MedRec:Name=MyServer,Type=Server" -property ListenPort

---------------------------
MBeanName: "MedRec:Name=MyServer,Type=Server"
        ListenPort: 7010

D:\>echo %ERRORLEVEL%
0
```

`weblogic.Admin` calls `System.exit(1)` if an exception is raised while processing a command, causing Ant and other Java client JVMs to exit. You can override this default behavior by specifying `-noExit` for Ant tasks (`wlconfig`) and `-continueOnError` for `weblogic.Admin` batch operations (`BATCHUPDATE`).

Command for Storing User Credentials

For any weblogic.Admin command that connects to a WebLogic Server instance, you must provide user credentials. You can use the STOREUSERCONFIG command to encrypt the user credentials instead of passing credentials directly on the command line or storing unencrypted credentials in scripts. See "Specifying User Credentials" on page 1-12.

STOREUSERCONFIG

Creates a user-configuration file and an associated key file. The user-configuration file contains an encrypted username and password. The key file contains a secret key that is used to encrypt and decrypt the username and password.

Only the key file that originally encrypted the username and password can decrypt the values. If you lose the key file, you must create a new user-configuration and key file pair.

Caution: You must ensure that only authorized users can access the key file. Any user who accesses a valid user-configuration and key file pair gains the privileges of the encrypted username. To secure access to the key file, you can store the key file in a directory that provides read and write access only to authorized users, such as WebLogic Server administrators. Alternatively, you can write the key file to a removable medium, such as a floppy or CD, and lock the medium in a drawer when it is not being used.

Unlike other weblogic.Admin commands, the STOREUSERCONFIG command does not connect to a WebLogic Server instance. The data encryption and file creation are accomplished by the JVM in which the STOREUSERCONFIG command runs. Because it does not connect to a WebLogic Server instance, the command cannot verify that the username and password are valid WebLogic Server credentials.

Syntax

```
java weblogic.Admin
  -username username [-password password]
  [ -userconfigfile config-file ] [ -userkeyfile keyfile ]
  STOREUSERCONFIG
```

Argument	Definition
`-userconfigfile` *`config-file`*	Specifies a file pathname at which the `STOREUSERCONFIG` command creates a user-configuration file. The pathname can be absolute or relative to the directory from which you enter the command. If a file already exists at the specified pathname, the command overwrites the file with a new file that contains the newly encrypted username and password. If you do not specify this option, `STOREUSERCONFIG` does the following: ■ To determine the directory in which to create the user-configuration file, it uses the JVM's user-home directory. The default value varies depending on the SDK and type of operating system. See "Configuring the Default Path Name" on page 1-17. ■ To determine the file name, it prepends your operating-system username to the string `-WebLogicConfig.properties`. For example, *`username`*`-WebLogicConfig.properties`. You can use Java options to specify a different username. See "Configuring the Default Path Name" on page 1-17.
`-userkeyfile` *`keyfile`*	Specifies a file pathname at which the `STOREUSERCONFIG` command creates a key file. The pathname can be absolute or relative to the directory from which you enter the command. If a file already exists at the specified pathname, `STOREUSERCONFIG` uses the existing key file to encrypt the new user-configuration file. If you do not specify this option, `STOREUSERCONFIG` does the following: ■ To determine the directory in which to create the key file, it uses the JVM's user-home directory. The default value varies depending on the SDK and type of operating system. See "Configuring the Default Path Name" on page 1-17. ■ To determine the file name, it prepends your operating-system username to the string `-WebLogicKey.properties`. For example, *`username`*`-WebLogicKey.properties`. You can use Java options to specify a different username. See "Configuring the Default Path Name" on page 1-17.

Argument	Definition (Continued)
`-username` *`username`* `[-password` *`password`* `]`	Specifies the username and password to encrypt. The `STOREUSERCONFIG` command does **not** verify that the username and password are valid WebLogic Server user credentials. If you omit the `-password` *`password`* argument, `STOREUSERCONFIG` prompts you to enter a password.

Configuring the Default Path Name

If you do not specify the location in which to create and use a user-configuration file and key file, the `weblogic.Admin` and `weblogic.Deployer` utilities supply the following default values:

- *`user-home-directory`*`\`*`username`*`-WebLogicConfig.properties`
- *`user-home-directory`*`\`*`username`*`-WebLogicKey.properties`

Where *`user-home-directory`* is the home directory of the operating-system user account as determined by the JVM, and *`username`* is your operating-system username.

The value of the home directory varies depending on the SDK and type of operating system. For example, on UNIX, the home directory is usually "~*`username`*." On Windows, the home directory is usually "`C:\Documents and Settings\`*`username`*".

You can use the following Java options to specify values for *`user-home-directory`* and *`username`*:

- `-Duser.home=`*`pathname`* specifies the value of *`user-home-directory`*
- `-Duser.name=`*`usernanme`* specifies the value of *`username`*.

For example, the following command configures the user-home directory to be `c:\myHome` and the user name to be `wlAdmin`. The command will search for the following user-configuration file and user key file:

```
c:\myHome\wlAdmin-WebLogicConfig.properties
c:\myHome\wlAdmin-WebLogicKey.properties
```

```
java -Duser.home=c:\myHome -Duser.name=wlAdmin
weblogic.Admin COMMAND
```

Creating User-Configuration and Key Files

To create user-configuration and key files:

1. Use the `-username` *username* and `-password` *password* arguments to specify the username and password to be encrypted.
2. Specify the name and location of the user-configuration and key files by doing one of the following:
   - Use the `-userconfigfile` *config-file* and `-userkeyfile` *key-file* arguments:
     ```
     java weblogic.Admin -username username -password password
      -userconfigfile config-file -userkeyfile key-file
     STOREUSERCONFIG
     ```
   - Use the default behavior to create files named *user-home-directory*\username-WebLogicConfig.properties and *user-home-directory\username*-WebLogicKey.properties:
     ```
     java weblogic.Admin -username username -password password
     STOREUSERCONFIG
     ```
   - Use the `-Duser.home=`*directory* and `-Duser.name=`*username* Java options to create files named *directory*\username-WebLogicConfig.properties and *directory\username*-WebLogicKey.properties:
     ```
     java -Duser.home=directory -Duser.name=username
     weblogic.Admin -username username -password password
     STOREUSERCONFIG
     ```

You can change the name and location of a user-configuration file or a key file after you create them, as long as you use the two files as a pair.

Using a Single Key File for Multiple User-Configuration Files

To use one key file to encrypt multiple user-configuration files:

1. Create an initial user-configuration file and key file pair.

   For example, enter the following command:

   ```
   java weblogic.Admin -username username -password password
   -userconfigfile c:\AdminConfig -userkeyfile e:\myKeyFile
   STOREUSERCONFIG
   ```

2. When you create an additional user-configuration file, specify the existing key file.

   For example, enter the following command:

   ```
   java weblogic.Admin -username username -password password
   -userconfigfile c:\anotherConfigFile -userkeyfile e:\myKeyFile
   STOREUSERCONFIG
   ```

Examples

In the following example, a user who is logged in to a UNIX operating system as `joe` encrypts the username `wlAdmin` and password `wlPass`:

```
java weblogic.Admin -username wlAdmin -password wlPass
STOREUSERCONFIG
```

The command determines whether a key file named `~joe/joe-WebLogicKey.properties` exists. If such a file does not exist, it prompts the user to select `y` to confirm creating a key file. If the command succeeds, it creates two files:

```
~joe\joe-WebLogicConfig.properties
~joe\joe-WebLogicKey.properties
```

The file `joe-WebLogicConfig.properties` contains an encrypted version of the strings `wlAdmin` and `wlPass`. Any command that uses the `~joe\joe-WebLogicConfig.properties` file must specify the `~joe\joe-WebLogicKey.properties` key file.

In the following example, the user `joe` is a System Administrator who wants to create a user-configuration file for an operating-system account named `pat`. For the sake of convenience, `joe` wants to create the user-configuration file in `pat`'s home directory, which will simplify the syntax of the `weblogic.Admin` commands that `pat` invokes. For added security, only one key file exists at `joe`'s organization, and it is located on a removable hard drive.

To create a user configuration file in `pat`'s home directory that is encrypted and decrypted by a key file name `e:\myKeyFile`:

```
java -Duser.name=pat -Duser.home="C:\Documents and Settings\pat"
weblogic.Admin -username wlOperatorPat -password wlOperator1 -userkeyfile
e:\myKeyFile
STOREUSERCONFIG
```

A user who logs in to `pat`'s account can use the following syntax to invoke `weblogic.Admin` commands:

```
java weblogic.Admin -userkeyfile e:\myKeyFile COMMAND
```

For information on using user-configuration and key files, see "Specifying User Credentials" on page 1-12.

Commands for Managing the Server Life Cycle

Table 1-4 is an overview of commands that manage the life cycle of a server instance. Subsequent sections describe command syntax and arguments, and provide an example for each command.

For more information about the life cycle of a server instance, refer to "Server Life Cycle" in the *Configuring and Managing WebLogic Server* guide.

Table 1-4 Overview of Commands for Managing the Server Life Cycle

Command	Description
CANCEL_SHUTDOWN	(Deprecated) Cancels the SHUTDOWN command for the WebLogic Server that is specified in the URL. See "CANCEL_SHUTDOWN" on page 1-21.
DISCOVERMANAGEDSERVER	Causes the Administration Server to re-establish its administrative control over Managed servers. See "DISCOVERMANAGEDSERVER" on page 1-21.
FORCESHUTDOWN	Terminates a server instance without waiting for active sessions to complete See "FORCESHUTDOWN" on page 1-24.
LOCK	(Deprecated) Locks a WebLogic Server against non-privileged logins. Any subsequent login attempt initiates a security exception which may contain an optional string message. See "LOCK" on page 1-28.
RESUME	Makes a server available to receive requests from external clients. See "RESUME" on page 1-29.
SHUTDOWN	Gracefully shuts down a WebLogic Server. See "SHUTDOWN" on page 1-30.
START	Uses a configured Node Manager to start a Managed Server in the RUNNING state. See "START" on page 1-35.
STARTINSTANDBY	(Deprecated) Uses a configured Node Manager to start a Managed Server and place it in the STANDBY state. See "STARTINSTANDBY" on page 1-37.
UNLOCK	(Deprecated) Unlocks the specified WebLogic Server after a LOCK operation. See "UNLOCK" on page 1-39.

CANCEL_SHUTDOWN

(Deprecated) The CANCEL_SHUTDOWN command cancels the SHUTDOWN command for a specified WebLogic Server.

When you use the SHUTDOWN 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 1-30 and "UNLOCK" on page 1-39.

This command is deprecated because the ability to specify a delay in the SHUTDOWN command is also deprecated. Instead of specifying a delay in the SHUTDOWN command, you can now set attributes to control how a server shuts down. For more information, refer to "Controlling Graceful Shutdowns" in the Administration Console Online Help.

Syntax

```
java [ SSL Arguments ] weblogic.Admin
     [ Connection Arguments ]
     [ User Credentials Arguments ]
     CANCEL_SHUTDOWN
```

Example

The following example cancels the shutdown of a WebLogic Server instance that runs on a machine named ManagedHost and listens on port 8001:

```
java weblogic.Admin -url ManagedHost:8001 -username weblogic
   -password weblogic CANCEL_SHUTDOWN
```

For more information about the environment in which this example runs, refer to "Example Environment" on page 1-14.

DISCOVERMANAGEDSERVER

Causes the Administration Server to re-establish administrative control over Managed Servers.

If the Administration Server fails while Managed Servers continue to run, or if you shut down the Administration Server while Managed Servers continue to run, you lose the ability to change the configuration or deploy modules to any server in the domain. To regain this administrative ability, you must restart the Administration Server. By default, an Administration Server finds the last known set of Managed Servers and re-establishes a connection.

If the Administration Server is unable to automatically re-establish a connection to one or more Managed Servers during its startup cycle, you can use this command to re-establish administrative control.

For example, you might have started a Managed Server in the `STANDBY` state and did not resume the Managed Server before restarting the Administration Server. The Administration Server discovers only Managed Servers that are in the `RUNNING` state.

Other factors can prevent the Administration Server from finding and re-connecting to Managed Servers, and you can use this command any time you need to re-establish a connection.

Syntax

```
java [ SSL Arguments ] weblogic.Admin
     [ -url [protocol://]listen-address:listen-port ]
     [ User Credentials Arguments ]
     DISCOVERMANAGEDSERVER [-serverName targetServer
     [-listenAddress listenaddress] [-listenPort listenport]
     [-listenPortSecure]]
```

Argument	Definition
`-url [protocol://]listen-address:listen-port`	You must specify the listen address and listen port of the Administration Server. If you specify a secure listen port, you must also specify a secure protocol. If you do not specify a value, the command assumes `t3://localhost:7001`. For more information, refer to the `-url` entry in Table 1-3 on page 11 and "Protocol Support" on page 1-13.
`-serverName`	Specifies a Managed Server that is currently running. If you do not specify a server, the Administration Server will discover and re-establish control over all the Managed Servers that are known to be running but disconnected from administrative services.

Argument	Definition (Continued)
`-listenAddress`	Specifies the listen address of the Managed Server that you name with the `-serverName` argument. If you do not specify this argument, the command uses the listen address that is configured in the domain's `config.xml` file.
`-listenPort`	Specifies the listen port of the Managed Server that you name with the `-serverName` argument. If you do not specify this argument, the command uses the listen port that is configured in the domain's `config.xml` file.
`-listenPortSecure`	Forces the Administration Server to use a secure protocol. Without this option, the Administration Server uses the t3 protocol. If you disable a Managed Server's non-SSL listen port, you must specify this option.

Example

The following command instructs the Administration Server to re-connect to MedRecManagedServer:

```
java weblogic.Admin -url AdminHost:7001 -username weblogic
   -password weblogic DISCOVERMANAGEDSERVER
   -serverName MedRecManagedServer
```

For more information about the environment in which this example runs, refer to "Example Environment" on page 1-14.

In the following example, a Managed Server is configured to listen on port 7021, but you used the `-Dweblogic.ListenPort` startup option to temporarily change the listen port to `8201`. The following command instructs the Administration Server to re-connect to MedRecManagedServer, which is listening on port 8201:

```
java weblogic.Admin -url AdminHost:7001 -username weblogic
   -password weblogic DISCOVERMANAGEDSERVER
   -serverName MedRecManagedServer -listenPort 8201
```

FORCESHUTDOWN

Terminates a server instance without waiting for active sessions to complete. For more information, refer to “Forced Shutdown" in the *Configuring and Managing WebLogic Server* guide.

If a server instance is in a deadlocked state, it can respond to `weblogic.Admin` commands only if you have enabled the domain-wide administration port. (A deadlocked server is one in which all threads are struck trying to acquire locks held by other threads.) If you have not already enabled the domain-wide administration port, your only option for shutting down the server instance is to kill the Java process that is running the server. You will lose all session data. For information on enabling the domain-wide administration port, refer to "Enabling the Domain-Wide Administration Port" in the Administration Console Online Help.

Syntax

```
java [ SSL Arguments ]
   [-Dweblogic.system.BootIdentityFile=filename
     [-Dweblogic.RootDirectory=path]
   ]
   weblogic.Admin
   [ Connection Arguments ]
   [ User Credentials Arguments ]
   FORCESHUTDOWN [targetServer]
```

Argument	Definition
`-Dweblogic.system.BootIdentityFile=`*`filename`* `[-Dweblogic.RootDirectory=`*`path`*`]`	Cause the command to retrieve encrypted user credentials from a boot identity file. See "Boot Identity Files" in the Administration Console Online Help. Use these arguments if you invoke this command from a script, you have not created a user configuration file, and you do not want to store user credentials in your script. If you do you not use the `-username` argument or a user configuration file to specify credentials (see "User Credentials Arguments" on page 1-10), the command retrieves user credentials from a boot properteis file as follows: • If you invoke the command from a server's root directory, and if the server's root directory contains a valid `boot.properties` file, it retrieves credentials from this file by default. For information about a server's root directory, refer to "A Server's Root Directory." • If you invoke the command from a server's root directory, but the server's boot identity file is not in the server's root directory or is not named `boot.properties`, the command can use a boot identity file if you include the following argument: `-Dweblogic.system.BootIdentityFile=`*`filename`* where *`filename`* is the fully qualified pathname of a valid boot identity file. • If you do not invoke the command from a server's root directory, the command can use a boot identity file if you include both of the following arguments in the command: `-Dweblogic.system.BootIdentityFile=`*`filename`* `-Dweblogic.RootDirectory=`*`path`* where *`filename`* is the fully qualified pathname of a valid boot identity file and `path` is the relative or fully-qualified name of the server's root directory. • If you have not created a boot identity file for a server, or if you do not want to use it, you must use the `-username` and `-password` arguments to provide user credentials. • If you specify both `-Dweblogic.system.BootIdentityFile=`*`filename`* and `-username` and `-password`, the command uses the credentials specified in the `-username` and `-password` arguments.

Argument	Definition (Continued)
targetServer	The name of the server to shut down. If you do not specify a value, the command shuts down the server that you specified in the -url argument.

Example

The following command instructs the Administration Server to shut down a Managed Server:

```
java weblogic.Admin -url AdminHost:7001 -username weblogic
   -password weblogic FORCESHUTDOWN MedRecManagedServer
```

After you issue the command, MedRecManagedServer prints messages to its log file and to its standard out. The messages indicate that the server state is changing and that the shutdown sequence is starting.

If the command succeeds, the final message that the target server prints is as follows:

```
<Oct 12, 2002 11:28:59 AM EDT> <Alert> <WebLogicServer> <000219> <The
shutdown sequence has been initiated.>
```

In addition, if the command succeeds, the weblogic.Admin utility returns the following:

```
Server "MedRecManagedServer" was force shutdown successfully ...
```

For more information about the environment in which this example runs, refer to "Example Environment" on page 1-14.

In the following example, the Administration Server is not available, so the command instructs the Managed Server to shut itself down:

```
java weblogic.Admin -url ManagedHost:8001 -username weblogic
-password weblogic FORCESHUTDOWN
```

The following example provides user credentials by referring to a boot identity file. The example specifies the server's root directory and boot identity file name so that it can be invoked from any directory:

```
java -Dweblogic.system.BootIdentityFile=c:\mydomain\boot.properties
 -Dweblogic.RootDirectory=c:\mydomain
 weblogic.Admin -url AdminHost:7001 FORCESHUTDOWN
```

LOCK

(Deprecated) Locks a WebLogic Server instance against non-privileged logins. Any subsequent login attempt initiates a security exception which may contain an optional string message.

Note: This command is privileged. It requires the password for the WebLogic Server administrative user.

Instead of using the `LOCK` command, start a server in the `STANDBY` state. In this state, a server instance responds only to administrative requests over the domain-wide administration port. See "Server Life Cycle" in *Configuring and Managing WebLogic Server*.

Syntax

```
java [ SSL Arguments ]
     weblogic.Admin
     [ -url [protocol://]listen-address:listen-port ]
     [ User Credentials Arguments ]
   LOCK ["stringMessage"]
```

Argument	Definition
`-url [protocol://]listen-address:listen-port`	Specify the listen address and listen port of the server instance that you want to lock. If you specify a secure listen port, you must also specify a secure protocol. If you do not specify a value, the command assumes `t3://localhost:7001`. For more information, refer to the `-url` entry in Table 1-3 on page 11 and "Protocol Support" on page 1-13.
"stringMessage"	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 instance is locked.

Example

In the following example, a Managed Server named MedRecManagedServer is locked.

```
java weblogic.Admin -url ManagedHost:8001 -username weblogic
   -password weblogic
   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.`

RESUME

Moves a server instance from the `STANDBY` state to the `RUNNING` state.

For more information about server states, refer to "Server Life Cycle" in the *Configuring and Managing WebLogic Server* guide.

Syntax

```
java [ SSL Arguments ]
     weblogic.Admin
     [-url URL]
     [ User Credentials Arguments ]
   RESUME [targetServer]
```

Argument	Definition
`-url` *`secure-protocol://listen-address:listen-port`*	Because servers can be in the `STANDBY` state only if the domain-wide administration port is enabled, to resume a server you must specify the Administration Server and domain-wide administration port as follows: `t3s://`*`Admin-Server-listen-address:domain-wide-admin-port`* or `https://`*`Admin-Server-listen-address:domain-wide-admin-port`* For more information, refer to the `-url` entry in Table 1-3 on page 11 and "Protocol Support" on page 1-13.
`targetServer`	The name of the server to resume. If you do not specify a value, the command resumes the server that you specified in the `-url` argument.

Example

The following example connects to the Administration Server and instructs it to resume a Managed Server:

```
java weblogic.Admin -url t3s://AdminHost:9002 -username weblogic
-password weblogic RESUME MedRecManagedServer
```

For more information about the environment in which this example runs, refer to “Example Environment” on page 1-14.

SHUTDOWN

Gracefully shuts down the specified WebLogic Server instance.

A graceful shutdown gives WebLogic Server subsystems time to complete certain application processing currently in progress. By default, a server instance waits for pending HTTP sessions to finish as a part of the graceful shutdown. You can override this behavior using the `-ignoreExistingSessions` argument. See “Graceful Shutdown" in the *Configuring and Managing WebLogic Server* guide.

In release 6.x, this command included an option to specify a number of seconds to wait before starting the shutdown process. This option is now deprecated. To support this deprecated option, this command assumes that a numerical value in the field immediately after the `SHUTDOWN` command indicates seconds. Thus, you cannot use this command to gracefully shut down a server whose name is made up entirely of numbers. Instead, you must use the Administration Console. For information, refer to "Shutting Down a Server" in the Administration Console Online Help.

Instead of specifying a delay in the `SHUTDOWN` command, you can now use a `-timeout` option, or set attributes in the Administration Console to control how a server shuts down. For more information, refer to "Controlling Graceful Shutdowns" in the Administration Console Online Help.

If a server instance is in a deadlocked state, it can respond to `weblogic.Admin` commands only if you have enabled the domain-wide administration port. (A deadlocked server is one in which all threads are struck trying to acquire locks held by other threads.) If you have not already enabled the domain-wide administration port, your only option for shutting down the server instance is to kill the Java process that is running the server. You will lose all session data. For information on enabling the domain-wide administration port, refer to "Enabling the Domain-Wide Administration Port" in the Administration Console Online Help.

Syntax

```
java [ SSL Arguments ]
     [-Dweblogic.system.BootIdentityFile=filename
       [-Dweblogic.RootDirectory=path]
     ]
     weblogic.Admin
     [ Connection Arguments ]
     [ User Credentials Arguments ]
   SHUTDOWN [-ignoreExistingSessions] [-timeout seconds]
   [targetServer]

(Deprecated)java [ SSL Arguments ]
     [-Dweblogic.system.BootIdentityFile=filename
       [-Dweblogic.RootDirectory=path]
     ]
     weblogic.Admin
     [ Connection Arguments ]
     [ User Credentials Arguments ]
   SHUTDOWN [seconds [“stringMessage”]] [targetServer]
```

Argument	Definition
-Dweblogic.system.BootIdentityFile=*filename* [-Dweblogic.RootDirectory=*path*]	Cause the command to retrieve encrypted user credentials from a boot identity file. See "Boot Identity Files" in the Administration Console Online Help. Use these arguments if you invoke this command from a script, you have not created a user configuration file, and you do not want to store user credentials in your script. If you do you not use the -username argument or a user configuration file to specify credentials (see "User Credentials Arguments" on page 1-10), the command retrieves user credentials from a boot properteis file as follows: • If you invoke the command from a server's root directory, and if the server's root directory contains a valid boot.properties file, it retrieves credentials from this file by default. For information about a server's root directory, refer to "A Server's Root Directory." • If you invoke the command from a server's root directory, but the server's boot identity file is not in the server's root directory or is not named boot.properties, the command can use a boot identity file if you include the following argument: -Dweblogic.system.BootIdentityFile=*filename* where *filename* is the fully qualified pathname of a valid boot identity file. • If you do not invoke the command from a server's root directory, the command can use a boot identity file if you include both of the following arguments in the command: -Dweblogic.system.BootIdentityFile=*filename* -Dweblogic.RootDirectory=*path* where *filename* is the fully qualified pathname of a valid boot identity file and path is the relative or fully-qualified name of the server's root directory. • If you have not created a boot identity file for a server, or if you do not want to use it, you must use the -username and -password arguments to provide user credentials. • If you specify both -Dweblogic.system.BootIdentityFile=*filename* and -username and -password, the command uses the credentials specified in the -username and -password arguments.

Argument	Definition
`-ignoreExistingSessions`	Causes a graceful shutdown operation to drop all HTTP sessions immediately. If you do not specify this option, the command refers to the Ignore Sessions During Shutdown setting for the server in the domain's `config.xml` file. For more information, refer to "Controlling Graceful Shutdowns" in the Administration Console Online Help. By default, a graceful shutdown operation waits for HTTP sessions to complete or timeout.
`-timeout` *`seconds`*	The number of seconds subsystems have to complete in-flight work and suspend themselves. If you specify a timeout value and the subsystems do not complete work and suspend themselves within that period, WebLogic Server will perform a forced shutdown on the server instance. If you do not specify a value, the `SHUTDOWN` command defers to the Graceful Shutdown Timeout setting that is specified on the Server→Control→Start/Stop tab of the Administration Console. By default, this setting is `0`, which means that the server will wait indefinitely for graceful shutdown to complete.
`targetServer`	The name of the server to shut down. If you do not specify a value, the command shuts down the server that you specified in the `-url` argument.
`seconds`	(Deprecated) Number of seconds allowed to elapse between the invoking of this command and the shutdown of the server.
`"stringMessage"`	(Deprecated) 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 being shut down.

Example

The following example instructs the Administration Server to shut down a Managed Server:

```
java weblogic.Admin -url AdminHost:7001 -username weblogic
-password weblogic SHUTDOWN MedRecManagedServer
```

After you issue the command, MedRecManagedServer prints messages to its log file and to its standard out. The messages indicate that the server state is changing and that the shutdown sequence is starting.

If the command succeeds, the final message that the target server prints is as follows:

```
<Oct 12, 2002 11:28:59 AM EDT> <Alert> <WebLogicServer> <000219> <The
shutdown sequence has been initiated.>
```

In addition, if the command succeeds, the `weblogic.Admin` utility returns the following:

```
Server "MedRecManagedServer" was shutdown successfully ...
```

For more information about the environment in which this example runs, refer to "Example Environment" on page 1-14.

In the following example, the Administration Server is not available. The same user connects to a Managed Server and instructs it to shut itself down:

```
java weblogic.Admin -url ManagedHost:8001 -username weblogic
-password weblogic SHUTDOWN
```

The following example provides user credentials by referring to a boot identity file. The example specifies the server's root directory and boot identity file name so that it can be invoked from any directory:

```
java -Dweblogic.system.BootIdentityFile=c:\mydomain\boot.properties
 -Dweblogic.RootDirectory=c:\mydomain weblogic.Admin
 -url AdminHost:7001 SHUTDOWN
```

START

Starts a Managed Server using Node Manager.

This command requires the following environment:

- The domain's Administration Server must be running.
- The Node Manager must be running on the Managed Server's host machine.
- The Managed Server must be configured to communicate with a Node Manager. For more information, refer to "Configuring a Machine" in the Administration Console Online Help.

The Startup Mode field in the Administration Console determines whether a Managed Server starts in the `RUNNING` state or `STANDBY` state. See "Server Life Cycle" in *Configuring and Managing WebLogic Server*.

Syntax

```
java [ SSL Arguments ] weblogic.Admin
    [-url URL]
    [ User Credentials Arguments ]
    START targetServer
```

Argument	Definition
`-url [protocol://]listen-address:listen-port`	Must specify the listen address and listen port of the domain's Administration Server. If you specify a secure listen port, you must also specify a secure protocol. If you do not specify a value, the command assumes `t3://localhost:7001`. For more information, refer to the `-url` entry in Table 1-3 on page 11 and "Protocol Support" on page 1-13.
targetServer	The name of the Managed Server to start in a `RUNNING` state.

Example

The following example instructs the Administration Server and Node Manager to start a Managed Server:

```
java weblogic.Admin -url AdminHost:7001 -username weblogic
-password weblogic START MedRecManagedServer
```

When you issue the command, the following occurs:

1. The Administration Server determines which machine `MedRecManagedServer` is configured to run on. It instructs the Node Manager that is running on that machine to start `MedRecManagedServer` in the state that the Start Mode field specifies.

2. The Node Manager indicates its progress by writing messages to its standard out. You can view these messages from the Administration Console on the Server→Control→Remote Start Output tab.

3. If the command succeeds, the `weblogic.Admin` utility returns to the following message:

```
Server "MedRecManagedServer" was started ...
Please refer to server log files for completion status ...
```

For more information about the environment in which this example runs, refer to "Example Environment" on page 1-14.

STARTINSTANDBY

(Deprecated) Starts a Managed Server using Node Manager.

In previous releases, this command started a Managed Server using the Node Manager **and** placed it in a `STANDBY` state. In this state, a server is not accessible to requests from external clients.

In the current release, the Startup Mode field in the Administration Console determines the state in which a Managed Server starts, regardless of which command you use to start the server instance. See "Starting a Managed Server in the STANDBY State" in the Administration Console Help.

This command requires the following environment:

- The domain's Administration Server must be running.
- The Node Manager must be running on the Managed Server's host machine.
- The Managed Server must be configured to communicate with a Node Manager. For more information, refer to "Configuring a Machine" in the Administration Console Online Help.
- The domain must be configured to use a domain-wide administration port as described in "Enabling the Domain-Wide Administration Port" in the Administration Console Online Help.

For more information about server states, refer to "Server Life Cycle" in the *Configuring and Managing WebLogic Server* guide.

Syntax

```
java [ SSL Arguments ] weblogic.Admin
    [-url URL]
    [ User Credentials Arguments ]
    STARTINSTANDBY targetServer
```

Argument	Definition
`-url` *`secure-protocol://listen-address:listen-port`*	You must specify the Administration Server and domain-wide administration port as follows: `t3s://`*`Admin-Server-listen-address:domain-wide-admin-port`* or `https://`*`Admin-Server-listen-address:domain-wide-admin-port`* For more information, refer to the `-url` entry in Table 1-3 on page 11 and "Protocol Support" on page 1-13.
`targetServer`	The name of the WebLogic Server to start in the `STANDBY` state.

Example

The following example instructs the Administration Server and Node Manager to start a Managed Server:

```
java weblogic.Admin -url t3s://AdminHost:9002 -username weblogic
-password weblogic STARTINSTANDBY MedRecManagedServer
```

When you issue the command, the following occurs:

1. The Administration Server determines which machine `MedRecManagedServer` is configured to run on. It instructs the Node Manager that is running on that machine to start `MedRecManagedServer` in the state that the Start Mode field specifies.

2. The Node Manager indicates its progress by writing messages to its standard out. You can view these messages from the Administration Console on the Server→Control→Remote Start Output tab.

3. If the command succeeds, the `weblogic.Admin` utility returns to the following message:

```
Server "MedRecManagedServer" was started ...
Please refer to server log files for completion status ...
```

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

For more information about the environment in which this example runs, refer to "Example Environment" on page 1-14.

UNLOCK

(Deprecated) Unlocks the specified WebLogic Server after a LOCK operation.

This command is deprecated because the LOCK command is deprecated. Instead of LOCK and UNLOCK, use STARTINSTANDY and RESUME. For more information, refer to "RESUME" on page 1-29.

Syntax

```
java [ SSL Arguments ] weblogic.Admin
     [-url URL]
     [ User Credentials Arguments ]
   UNLOCK
```

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
```

Commands for Retrieving Information about WebLogic Server and Server Instances

Table 1-5 is an overview of commands that return information about WebLogic Server installations and instances of WebLogic Server. Subsequent sections describe command syntax and arguments, and provide an example for each command.

Table 1-5 Overview of Commands for Retrieving Information about WebLogic Server

Command	Description
CONNECT	Makes the specified number of connections to a WebLogic Server instance 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 1-40.
GETSTATE	Returns the current state of the specified WebLogic Server instance. See "GETSTATE" on page 1-42.

Table 1-5 Overview of Commands for Retrieving Information about WebLogic Server (Continued)

Command	Description
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 1-43.
LICENSES	Lists the licenses for all WebLogic Server instances that are installed on a specific server. See "LICENSES" on page 1-44.
LIST	Lists the bindings of a node in a server's JNDI naming tree. See "LIST" on page 1-44.
PING	Sends a message to verify that a WebLogic Server instance is listening on a port and is ready to accept client requests. See "PING" on page 1-46. For a similar command that returns information about all servers in a cluster, see "CLUSTERSTATE" on page 1-81.
SERVERLOG	Displays the server log file generated on a specific server instance. See "SERVERLOG" on page 1-47.
THREAD_DUMP	Provides a real-time snapshot of the WebLogic Server threads that are currently running on a particular instance. See "THREAD_DUMP" on page 1-49.
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 1-50.

CONNECT

Connects to a WebLogic Server instance 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 [ SSL Arguments ] weblogic.Admin
     [-url URL]
     [ User Credentials Arguments ]
   CONNECT [count]
```

Argument	Definition
`-url [protocol://]listen-address:listen-port`	Specify the listen address and listen port of the server instance to which you want to connect. If you specify a secure listen port, you must also specify a secure protocol. If you do not specify a value, the command assumes `t3://localhost:7001`. For more information, refer to the `-url` entry in Table 1-3 on page 11 and "Protocol Support" on page 1-13.
count	Number of connections the `weblogic.Admin` utility makes to the specified server instance. By default, this command makes only one connection.

Example

In the following example, the `weblogic.Admin` utility establishes 10 connections to a WebLogic Server instance whose listen address is ManagedHost and listen port is 8001:

```
java weblogic.Admin -url ManagedHost:8001 -username weblogic
   -password weblogic CONNECT 10
```

For more information about the environment in which this example runs, refer to "Example Environment" on page 1-14.

If the command establishes the connections, it returns the following information:

```
Connection: 0 - 3,229 ms
Connection: 1 - 17 ms
Connection: 2 - 14 ms
Connection: 3 - 20 ms
Connection: 4 - 18 ms
Connection: 5 - 25 ms
Connection: 6 - 27 ms
```

```
Connection: 7 - 15 ms
Connection: 8 - 15 ms
Connection: 9 - 15 ms
  RTT = ~3422 milliseconds, or ~342 milliseconds/connection
```

If the command does not establish a connection, it returns nothing.

In this example, the first connection required 3,229 milliseconds and the second connection required 17 milliseconds. The average time for all connections was 3422 milliseconds.

GETSTATE

Returns the current state of a server.

For more information about server states, refer to "Server Life Cycle" in the *Configuring and Managing WebLogic Server* guide.

If a server instance is in a deadlocked state, it can respond to `weblogic.Admin` commands only if you have enabled the domain-wide administration port. (A deadlocked server is one in which all threads are struck trying to acquire locks held by other threads.) If you have not already enabled the domain-wide administration port, your only option is to shut down the server instance by killing the Java process that is running the server. You will lose all session data. For information on enabling the domain-wide administration port, refer to "Enabling the Domain-Wide Administration Port" in the Administration Console Online Help.

Syntax

```
java [ SSL Arguments ] weblogic.Admin
     [ Connection Arguments ]
     [ User Credentials Arguments ]
   GETSTATE [targetServer]
```

Argument	Definition
targetServer	The name of the server for which you want to retrieve the current state. If you do not specify a value, the command returns the state of the server that you specified in the `-url` argument.

Example

The following example returns the state of a WebLogic Server instance that runs on a machine named AdminHost:

```
java weblogic.Admin -url AdminHost:7001 -username weblogic
   -password weblogic GETSTATE
```

For more information about the environment in which this example runs, refer to "Example Environment" on page 1-14.

If the command succeeds for a running server, it returns the following:

```
Current state of "MedRecServer" : RUNNING
```

For a complete list of server states, refer to "Server Life Cycle" in the *Configuring and Managing WebLogic Server* guide.

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.

You can issue this command from any computer on which the WebLogic Server is installed. You do not need to start a server instance to invoke this command, nor do you need to supply user credentials.

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 command returns the following:

```
Usage: java [SSL trust options]
    weblogic.Admin [ [-url | -adminurl] [<protocol>://]<listen-address>:<port>]
    -username <username> -password <password>
    PING <roundTrips> <messageLength>

Where:
roundTrips = Number of pings.
messageLength = Size of the packet (in bytes) to send in each ping. The default
```

```
size is 100 bytes. Requests for pings with packets larger than 10 MB throw
exceptions.

Description: Sends a message to verify that a WebLogic Server instance is listen
ing on a port and is ready to accept WebLogic client requests.

Example(s):
Connecting through a non-secured port:
java weblogic.Admin -url t3://localhost:7001 -username weblogic -password weblog
ic ping 3 100

Connecting through an SSL port of a server that uses the demonstration keys and
certificates:
|java -Dweblogic.security.TrustKeyStore=DemoTrust
weblogic.Admin -url t3s:\localhost:7001 -username weblogic -password weblogic
PING <roundTrips> <messageLength>
```

LICENSES

Lists the BEA licenses for all WebLogic Server instances installed on the specified host.

Syntax

```
java [ SSL Arguments ] weblogic.Admin
     [ Connection Arguments ]
     [ User Credentials Arguments ]
   LICENSES
```

Example

The following command returns a list of licenses for a host named AdminHost:

```
java weblogic.Admin -url AdminHost:7001 -username weblogic
   -password weblogic LICENSES
```

For more information about the environment in which this example runs, refer to "Example Environment" on page 1-14.

If the command establishes a connection, it returns license information to standard out.

LIST

Lists the bindings of a node in the JNDI naming tree.

Syntax

```
java [ SSL Arguments ] weblogic.Admin
     [ Connection Arguments ]
     [ User Credentials Arguments ]
   LIST [JNDIcontextName]
```

Argument	Definition
JNDIcontextName	The JNDI context for lookup, for example, `weblogic`, `weblogic.ejb`, `javax`. By default, the command lists the bindings immediately below the InitialContext of the specified server instance.

Example

The following command returns the initial context for the MedRecServer example server that runs on a machine named AdminHost:

```
java weblogic.Admin -url AdminHost:7001 -username weblogic
   -password weblogic LIST
```

For more information about the environment in which this example runs, refer to "Example Environment" on page 1-14.

If the command succeeds, it returns information similar to the following abbreviated output:

```
Contents of InitialContext
    jms: weblogic.jndi.internal.ServerNamingNode
  javax: weblogic.jndi.internal.ServerNamingNode
  mail: weblogic.jndi.internal.ServerNamingNode
...
```

To view the JNDI tree below the mail context, enter the following command:

```
 java weblogic.Admin -url AdminHost:7001 -username weblogic
   -password weblogic LIST mail
```

If the command succeeds, it returns the following:

```
Contents of mail
  MedRecMailSession: javax.mail.Session
```

PING

Sends a message to verify that a WebLogic Server instance is listening on a port and is ready to accept WebLogic client requests.

For information on returning a description of all servers in a cluster, refer to "CLUSTERSTATE" on page 1-81.

If a server instance is in a deadlocked state, it can respond to `weblogic.Admin` commands only if you have enabled the domain-wide administration port. (A deadlocked server is one in which all threads are struck trying to acquire locks held by other threads.) If you have not already enabled the domain-wide administration port, your only option is to shut down the server instance by killing the Java process that is running the server. You will lose all session data. For information on enabling the domain-wide administration port, refer to "Enabling the Domain-Wide Administration Port" in the Administration Console Online Help.

Syntax

```
java [ SSL Arguments ] weblogic.Admin
     [ Connection Arguments ]
     [ User Credentials Arguments ]
   PING [roundTrips] [messageLength]
```

Argument	Definition
roundTrips	Number of pings.
messageLength	Size of the packet (in bytes) to be sent in each ping. Requests for pings with packets larger than 10 MB throw exceptions.

Example

The following command pings a server instance 10 times:

```
java weblogic.Admin -url AdminHost:7001 -username weblogic
   -password weblogic PING 10
```

For more information about the environment in which this example runs, refer to "Example Environment" on page 1-14.

If the command succeeds, it returns output similar to the following:

```
Sending 10 pings of 100 bytes.
  RTT = ~46 milliseconds, or ~4 milliseconds/packet
```

The following command pings a server instance that is running on a host computer named ManagedHost:

```
java weblogic.Admin -url ManagedHost:8001 -username weblogic
   -password weblogic PING
```

SERVERLOG

Returns messages from the local log file of a server instance. The command returns messages only from the current log file; it does not return messages in log files that the server instance has archived (renamed) because of log file rotation.

By default, the command returns the first 500 messages from the current log file (messages within the file are ordered from oldest to newest). You can change the default behavior by specifying a time and date range, but you cannot change the number of messages to be returned. The command always returns up to 500 messages, depending on the number of messages in the log file.

For each message, the command returns the following message attributes, separated by spaces:

```
MessageID TimeStamp Severity Subsystem MessageText
```

For more information about message attributes, refer to "Message Attributes" in the Administration Console Online Help.

This command can not be used to return the domain-wide log file. You can view the domain-wide log file from the Administration Console. For more information about server log files, refer to "Local Log Files and Domain Log Files" in the Administration Console Online Help.

Syntax

```
 java [ SSL Arguments ] weblogic.Admin
      [-url URL]
      [ User Credentials Arguments ]
   SERVERLOG [starttime [endtime]]
```

Argument	Definition
`-url [protocol://]listen-address:listen-port`	Specify the listen address and listen port of the server instance for which you want to retrieve the local log file. If you use the `-url` argument to specify the Administration Server, the command returns the local log file of the Administration Server. If you specify a secure listen port, you must also specify a secure protocol. If you do not specify a value, the command assumes `t3://localhost:7001`. For more information, refer to the `-url` entry in Table 1-3 on page 11 and "Protocol Support" on page 1-13.
starttime	Returns up to 500 messages in the current log file with a time stamp that is **after** the time you specify. 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*" By default, `SERVERLOG` returns up to 500 messages in chronological order starting from the beginning of the current log file.
endtime	Specifies the end of a time range and causes `SERVERLOG` to return up to 500 messages with a time stamp that is after *starttime* and before *endtime*. 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*" By default, `SERVERLOG` returns up to 500 messages in chronological order starting with the *starttime* value and ending with the time at which you issued the `SERVERLOG` command.

Example

The following command returns all messages in the local log file of a server instance named MedRecManagedServer and pipes the output through the command shell's `more` command:

```
java weblogic.Admin -url ManagedHost:8001 -username weblogic
   -password weblogic SERVERLOG | more
```

For more information about the environment in which this example runs, refer to "Example Environment" on page 1-14.

If the command succeeds, it returns output similar to the following truncated example:

```
130036    Oct 18, 2002 4:19:12 PM EDT Info  XML   Initializing XMLRegistry.
001007    Oct 18, 2002 4:19:13 PM EDT  Info   JDBC  Initializing... issued.
001007    Oct 18, 2002 4:19:13 PM EDT  Info JDBC   Initialize Done issued.
190000    Oct 18, 2002 4:19:13 PM EDT   Info   Connector  Initializing J2EE
Connector Service
190001    Oct 18, 2002 4:19:13 PM EDT     Info     Connector J2EE Connector
Service initialized successfully
...
```

The following command returns messages that were written to the local log file since 8:00 am today:

```
java weblogic.Admin -url ManagedHost:8001 -username weblogic
   -password weblogic SERVERLOG 08:00
```

The following command returns messages that were written to the local log file between 8:00 am and 8:30 am on October 18, 2002:

```
java weblogic.Admin -url ManagedHost:8001 -username weblogic
 -password weblogic SERVERLOG "2002/10/18 08:00" "2002/10/18 08:30"
```

THREAD_DUMP

Prints a snapshot of the WebLogic Server threads that are currently running for a specific server instance. The server instance prints the snapshot to its standard out.

If a server instance is in a deadlocked state, it can respond to `weblogic.Admin` commands only if you have enabled the domain-wide administration port. (A deadlocked server is one in which all threads are struck trying to acquire locks held by other threads.) If you have not already enabled the domain-wide administration port, your only option is to shut down the server instance by killing the Java process that is running the server. You will lose all session data. For information on enabling the domain-wide administration port, refer to "Enabling the Domain-Wide Administration Port" in the Administration Console Online Help.

Note: The `THREAD_DUMP` command is supported only on Sun JVM and JRockit.

Syntax

```
java [ SSL Arguments ] weblogic.Admin
     [ Connection Arguments ]
     [ User Credentials Arguments ]
     THREAD_DUMP
```

Example

The following example causes a server instance that is running on a host named ManagedHost to print a thread dump to standard out:

```
java weblogic.Admin -url ManagedHost:8001 -username weblogic
-password weblogic THREAD_DUMP
```

If the command succeeds, the command itself returns the following:

```
Thread Dump is available in the command window that is running the server.
```

The server instance prints a thread dump to its standard out, which, by default, is the shell (command prompt) within which the server instance is running.

For more information about the environment in which this example runs, refer to "Example Environment" on page 1-14.

VERSION

Displays the version of the WebLogic Server software that is running the server instance you specify with the `-url` argument.

Syntax

```
java weblogic.Admin [-url URL] -username username
   [-password password] VERSION
```

Argument	Definition
`-url [protocol://]listen-address:listen-port`	Specify the listen address and listen port of a WebLogic Server instance. If you specify a secure listen port, you must also specify a secure protocol. If you do not specify a value, the command assumes `t3://localhost:7001`. For more information, refer to the `-url` entry in Table 1-3 on page 11 and "Protocol Support" on page 1-13.

Example

The following command displays the version of the WebLogic Server software that is currently running on a host named ManagedHost:

```
java weblogic.Admin -url ManagedHost:8001 -username weblogic
   -password weblogic VERSION
```

For more information about the environment in which this example runs, refer to "Example Environment" on page 1-14.

If the command succeeds, it returns output similar to the following:

```
WebLogic Server 8.1  Sat Oct 15 22:51:04 EDT 2002 207896
WebLogic XMLX Module 8.1  Sat Oct 15 22:51:04 EDT 2002 207896
```

Commands for Managing JDBC Connection Pools

Table 1-6 lists the WebLogic Server administration commands for connection pools. Subsequent 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/docs81/jdbc/index.html` and "JDBC Connection Pools" in the Administration Console Online Help.

Note: All JDBC commands are privileged commands. You must supply the username and password for a WebLogic Server administrative user to use these commands.

Table 1-6 Overview of Commands for Managing JDBC Connection Pools

Command	Description
`CREATE_POOL`	Creates a connection pool. See "CREATE_POOL" on page 1-53.
`DESTROY_POOL`	Closes all connections in the connection pool and deletes the connection pool configuration. See "DESTROY_POOL" on page 1-56.
`DISABLE_POOL`	Temporarily disables a connection pool, preventing any clients from obtaining a connection from the pool. See "DISABLE_POOL" on page 1-57.
`ENABLE_POOL`	Enables a connection pool 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 1-58.
`TEST_POOL`	Tests a connection pool by reserving and releasing a connection from the connection pool. Requires testConnsOnReserve or testConnsOnRelease to be true and testTableName must be set. See "TEST_POOL" on page 1-59

Table 1-6 Overview of Commands for Managing JDBC Connection Pools

Command	Description
`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 1-60.
`EXISTS_POOL`	Tests whether a connection pool with a specified name exists in a specified WebLogic Server instance. 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 1-61.

CREATE_POOL

Creates a connection pool on the WebLogic Server instance running at the specified URL. For more information, see "Creating a Connection Pool Dynamically" in *Programming WebLogic JDBC* at `http://e-docs.bea.com/wls/docs81/jdbc/programming.html#programming004`.

Syntax

```
java [ SSL Arguments ] weblogic.Admin
     [ Connection Arguments ]
     [ User Credentials Arguments ]
   CREATE_POOL poolName props=myProps,initialCapacity=1,maxCapacity=1,
   capacityIncrement=1,allowShrinking=true,shrinkPeriodMins=15,
   driver=myDriver,url=myURL, testConnsOnReserve=true,
   testTableName=tablename, testFrequencySeconds=0
```

Argument	Definition
`poolName`	Required. A unique name of the connection pool. Must be unique in the domain.
`aclName`	(Deprecated) Identifies the different access lists within `fileRealm.properties` in the server configuration directory. Paired name must be dynaPool.

Argument	Definition
props	Required. Database connection properties; typically in the format "database login name; server network id". Required entries vary by DBMS. Separate property value pairs with a semicolon.
password	Optional. Database login password. This value overrides any database password specified in props.
initialCapacity	Optional. Initial number of connections in a pool. If this property is defined, WebLogic Server creates these connections at boot time. Default is 1; cannot exceed maxCapacity.
maxCapacity	Optional. Maximum number of connections allowed in the pool. Default is 1; if defined, maxCapacity should be =>1.
capacityIncrement	Optional. Number of connections to add at a time when the connection pool is expanded. Default = 1.
allowShrinking	Optional. Indicates whether or not the pool can shrink when connections are detected to not be in use. Default = true.
shrinkPeriodMins	Optional. Interval between shrinking. Units in minutes. Minimum = 1.If `allowShrinking = True`, then default = 15 minutes.
driver	Required. Fully qualified driver classname of the JDBC driver.
url	Required. URL of the database as required by the JDBC driver. Format varies by DBMS.
testConnsOnReserve	Optional. Indicates reserved test connections. Default = False.
testConnsOnRelease	Optional. Indicates test connections when they are released. Default = False.
testTableName	Optional. Database table used when testing connections; must be present for tests to succeed. Required if either testConnsOnReserve or testConnsOnRelease are defined. Can also be a SQL query instead of a database table name. To use a SQL query, enter `SQL` followed by a space and the SQL query to run in place of the standard test.
refreshPeriod	Optional. Sets the connection refresh interval in minutes. 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.

Argument	Definition
`loginDelaySecs`	Optional. 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.
`testFrequencySeconds`	Optional. The number of seconds between database connection tests. After every interval specified by this value, unused database connections are tested using `testTableName`. Connections that do not pass the test are closed and reopened to re-establish a valid physical database connection. **Note:** You cannot test the database connection if `testTableName` is not set. When `testFrequencySeconds` is set to `0` (default value), periodic database testing is disabled.

Example

The following example creates a connection pool named demoPool on the MedRecManagedServer instance running on the host machine named ManagedHost and listening at port 7001 (when entered on a single command line):

```
java weblogic.Admin -url t3://ManagedHost:7001 -username weblogic
   -password weblogic CREATE_POOL demoPool
   url=jdbc:pointbase:server://localhost:9092/demo,
   driver=com.pointbase.jdbc.jdbcUniversalDriver,
   testConnsOnReserve=true,
   testTableName=SYSTABLES,
   initialCapacity=2,
   maxCapacity=10,
   capacityIncrement=2,
   allowShrinking=true,
   props=user=examples;password=examples
```

If the command succeeds, it returns output similar to the following:

```
Connection pool "demoPool" created successfully.
```

After creating a JDBC connection pool, you can create a data source or a transactional data source that applications can use to access the JDBC connection pool.

The following commands create a transactional data source named demoDS on the MedRecManagedServer instance:

```
java weblogic.Admin -url t3://ManagedHost:7001 -username weblogic -password
weblogic CREATE -mBean "medrec:Type=JDBCTxDataSource,Name=demoDS"
```

```
java weblogic.Admin -url t3://ManagedHost:7001 -username weblogic -password
weblogic SET -mBean "medrec:Name=demoDS,Type=JDBCTxDataSource" -property
"JNDIName" "demoDS" -property "PoolName" "demoPool"
```

DESTROY_POOL

Connections are closed and removed from the pool and the pool is destroyed when it has no remaining connections.

Syntax

```
java [ SSL Arguments ] weblogic.Admin
      [-url URL]
      [ User Credentials Arguments ]
   DESTROY_POOL poolName [true|false]
```

Argument	Definition
`-url [protocol://]listen-address:listen-port`	Specify the listen address and listen port of the Administration Server. If you specify a secure listen port, you must also specify a secure protocol. If you do not specify a value, the command assumes `t3://localhost:7001`. For more information, refer to the `-url` entry in Table 1-3 on page 11 and "Protocol Support" on page 1-13.
poolName	Required. Unique name of pool.

Argument	Definition
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

The following command destroys a connection pool named demoPool in a WebLogic Domain with the Administration Server running on a machine named AdminHost and listening at port 7001.

```
java weblogic.Admin -url AdminHost:7001 -username weblogic
   -password weblogic DESTROY_POOL demoPool false
```

DISABLE_POOL

You can temporarily disable a connection pool, preventing any clients from obtaining a connection from the pool.

You have the following options for disabling a pool. 1) Freeze the connections in a pool that you later plan to enable, or 2) Destroy the connections.

Syntax

```
java [ SSL Arguments ] weblogic.Admin
     [-url URL]
     [ User Credentials Arguments ]
   DISABLE_POOL -poolName connection_pool_name [true|false]
```

Argument	Definition
`-url [protocol://]listen-address:listen-port`	Optional. Specify the listen address and listen port of a WebLogic Server instance on which the connection pool has been deployed. If you specify a secure listen port, you must also specify a secure protocol. If you do not specify a value, the command assumes `t3://localhost:7001`. For more information, refer to the `-url` entry in Table 1-3 on page 11 and "Protocol Support" on page 1-13.
`-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, the command disables a connection pool named demoPool in a WebLogic domain where the Administration Server is running on a machine named AdminHost and listening at port 7001. This command freezes connections to be enabled later:

```
java weblogic.Admin -url AdminHost:7001 -username weblogic
   -password weblogic DISABLE_POOL -poolName demoPool 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 connection_pool_name
```

Argument	Definition
`-url [protocol://]listen-address:listen-port`	Optional. Specify the listen address and listen port of a WebLogic Server instance on which the connection pool has been deployed and is disabled. If you specify a secure listen port, you must also specify a secure protocol. If you do not specify a value, the command assumes `t3://localhost:7001`. For more information, refer to the `-url` entry in Table 1-3 on page 11 and "Protocol Support" on page 1-13.
`-poolName`	Name of the connection pool.

Example

In the following command, a connection pool named demoPool is re-enabled after being disabled:

```
java weblogic.Admin -url AdminHost:7001 -username weblogic
   -password weblogic ENABLE_POOL -poolName demoPool
```

TEST_POOL

Reserves and releases a connection from the connection pool. The command also runs a test query using the connection, either before reserving the connection or after releasing the connection, to make sure the database is available. This command requires that either testConnsOnReserve or testConnsOnRelease is set to true and testTableName is specified.

Note: The TEST_POOL command tests an individual instance of the connection pool. To test all instances (deployments) of the connection pool, repeat the command for each instance in your configuration.

Syntax

```
java [ SSL Arguments ] weblogic.Admin
     [-url URL]
     [ User Credentials Arguments ]
   TEST_POOL -poolName connection_pool_name
```

Argument	Definition
`-url [protocol://]listen-address:listen-port`	Specify the listen address and listen port of a WebLogic Server instance on which the connection pool has been deployed. Providing the url of a server on which the connection pool has NOT been deployed will return incorrect results. If the pool is deployed on multiple servers, run the command multiple times, each time pointing to one server instance on which the connection pool has been deployed. Running the command on only one server instance DOES NOT return the overall, aggregated status of the pool. See Note on page 59. If you specify a secure listen port, you must also specify a secure protocol. If you do not specify a value, the command assumes `t3://localhost:7001`. For more information, refer to the `-url` entry in Table 1-3 on page 11 and "Protocol Support" on page 1-13.
`-poolName`	Name of the connection pool as listed in the configuration file (`config.xml`).

Example

This command tests the connection pool registered as `MedRecPool` and deployed on a server that listens on port 8001 of the host ManagedHost:

```
java weblogic.Admin -url ManagedHost:8001 -username weblogic -password
weblogic TEST_POOL -poolName MedRecPool
```

If the command succeeds, it returns the following:

```
JDBC Connection Test Succeeded for connection pool "MedRecPool".
```

RESET_POOL

This command closes and reopens the database connections in a connection pool.

Syntax

```
java [ SSL Arguments ] weblogic.Admin
     [-url URL]
     [ User Credentials Arguments ]
   RESET_POOL -poolName connection_pool_name
```

Argument	Definition
`-url [protocol://]listen-address:listen-port`	Optional. Specify the listen address and listen port of a WebLogic Server instance on which the connection pool has been deployed. If you specify a secure listen port, you must also specify a secure protocol. If you do not specify a value, the command assumes `t3://localhost:7001`. For more information, refer to the `-url` entry in Table 1-3 on page 11 and "Protocol Support" on page 1-13.
`-poolName`	Name of the connection pool as listed in the configuration file (`config.xml`).

Example

This command closes and reopens database connection in the connection pool named demoPool for the WebLogic Server instance listening on port 7001 of the host AdminHost.

```
java weblogic.Admin -url AdminHost:7001 -username weblogic
  -password weblogic RESET_POOL -poolName demoPool
```

EXISTS_POOL

Tests whether a connection pool with a specified name exists in the WebLogic Server domain. 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 [ SSL Arguments ] weblogic.Admin
     [-url URL]
```

```
      [ User Credentials Arguments ]
    EXISTS_POOL -poolName connection_pool_name
```

Argument	Definition
-url [*protocol*://]*listen-address*:*listen-port*	Specify the listen address and listen port of the Administration Server. If you specify a secure listen port, you must also specify a secure protocol. If you do not specify a value, the command assumes `t3://localhost:7001`. For more information, refer to the `-url` entry in Table 1-3 on page 11 and "Protocol Support" on page 1-13.
-poolName	Name of connection pool.

Example

The following command determines wether or not a pool with a specific name exists:

```
java weblogic.Admin -url AdminHost:7001 -username weblogic
   -password weblogic EXISTS_POOL -poolName demoPool
```

Commands for Managing WebLogic Server MBeans

The following sections describe `weblogic.Admin` commands for managing WebLogic Server MBeans.

- "Specifying MBean Types" on page 1-62
- "MBean Management Commands" on page 1-63

Specifying MBean Types

To specify which MBean or MBeans you want to access, view, or modify, all of the MBean management commands require either the `-mbean` argument or the `-type` argument.

Use the `-mbean` argument to operate on a single instance of an MBean.

Use the `-type` argument to operate on all MBeans that are an instance of a type that you specify. An MBean's **type** refers to the interface class of which the MBean is an instance. All WebLogic Server MBeans are an instance of one of the interface classes defined in the `weblogic.management.configuration` or `weblogic.management.runtime` packages. For configuration MBeans, type also refers to whether an instance is an Administration MBean or a Local Configuration MBean. For a complete list of all WebLogic Server MBean interface classes, refer to the WebLogic Server Javadoc for the `weblogic.management.configuration` or `weblogic.management.runtime` packages.

To determine the value that you provide for the `-type` argument, do the following:

1. Find the MBean's interface class and remove the `MBean` suffix from the class name. For an MBean that is an instance of the `weblogic.management.runtime.JDBCConnectionPoolRuntimeMBean`, use `JDBCConnectionPoolRuntime`.
2. For a Local Configuration MBean, append `Config` to the name. For example, for a Local Configuration MBean that is an instance of the `weblogic.management.configuration.JDBCConnectionPoolMBean` interface class, use `JDBCConnectionPoolConfig`. For the corresponding Administration MBean instance, use `JDBCConnectionPool`.

MBean Management Commands

Table 1-7 is an overview of the MBean management commands.

Table 1-7 MBean Management Command Overview

Command	Description
`CREATE`	Creates an Administration MBean instance. This command cannot be used for Runtime MBeans and we recommend that you do not use it to create Local Configuration MBeans. See "CREATE" on page 1-64.
`DELETE`	Deletes an MBean instance. See "DELETE" on page 1-66.
`GET`	Displays properties of MBeans. See "GET" on page 1-68.
`INVOKE`	Invokes management operations that an MBean exposes for its underlying resource. See "INVOKE" on page 1-70.

Table 1-7 MBean Management Command Overview (Continued)

Command	Description
QUERY	Searches for MBeans whose WebLogicObjectName matches a pattern that you specify. See "QUERY" on page 1-72.
SET	Sets the specified property values for the named MBean instance. This command cannot be used for Runtime MBeans. See "SET" on page 1-75.

CREATE

Creates an instance of a WebLogic Server Administration or Local Configuration MBean, however, BEA recommends that you do not use it to create Local Configuration MBeans. This command cannot be used for Runtime MBeans.

If the command is successful, it returns OK.

When you use this command to create an Administration MBean instance, you must use the -url argument to specify the Administration Server. WebLogic Server populates the Administration MBean with default values and saves the MBean's configuration in the domain's config.xml file. For some types of Administration MBeans, WebLogic Server does not create the corresponding Local Configuration MBean replica until you restart the server instance that hosts the underlying managed resource. For example, if you create a JDBCConnectionPool Administration MBean to manage a JDBC connection pool on a Managed Server named ManagedMedRecServer, you must restart ManagedMedRecServer so that it can create its local replica of the JDBCConnectionPool Administration MBean that you created. For more information on MBean replication and the life cycle of MBeans, refer to "MBeans for Configuring Managed Resources" in the *Programming WebLogic Management Services with JMX* guide.

Syntax

```
java [ SSL Arguments ] weblogic.Admin
     [-url URL]
     [ User Credentials Arguments ]
   CREATE -name name -type mbeanType
```

or

```
java [ SSL Arguments ] weblogic.Admin
     [-url URL]
     [ User Credentials Arguments ]
   CREATE -mbean objectName
```

Argument	Definition
`-url [protocol://]listen-address:listen-port`	Specify the listen address and listen port of the Administration Server. You can create Administration MBeans only on the Administration Server. If you specify a secure listen port, you must also specify a secure protocol. If you do not specify a value, the command assumes `t3://localhost:7001`. For more information, refer to the `-url` entry in Table 1-3 on page 11 and "Protocol Support" on page 1-13. Although the `CREATE` command also supports the `-adminurl` argument, we recommend that you do not use `CREATE` to create Local Configuration MBeans.
`-name name`	The name you choose for the MBean that you are creating.
`-type mbeanType`	The type of MBean that you are creating. For more information, refer to "Specifying MBean Types" on page 1-62.
`-mbean objectName`	Fully qualified object name of an MBean in the `WebLogicObjectName` format. For example: `"domain:Type=type,Name=name"` For more information, refer to the Javadoc for `WebLogicObjectName`.

Example

The following example uses the `-name` and `-type` arguments to create a `JDBCConnectionPool` Administration MBean named `myPool` on an Administration Server:

```
java weblogic.Admin -url AdminHost:7001 -username weblogic
   -password weblogic CREATE -name myPool -type JDBCConnectionPool
```

For more information about the environment in which this example runs, refer to "Example Environment" on page 1-14.

If the command succeeds, it prints the following to standard out:

```
Ok
```

The following example uses the `-mbean` argument and `WebLogicObjectName` conventions to create a `JDBCConnectionPool` Administration MBean named `myPool` on an Administration Server:

```
java weblogic.Admin -url AdminHost:7001 -username weblogic
   -password weblogic
    CREATE -mbean “mydomain:Type=JDBCConnectionPool,Name=myPool”
```

DELETE

Deletes MBeans. If you delete an Administration MBean, WebLogic Server removes the corresponding entry from the domain’s `config.xml` file.

If the command is successful, it returns `OK`.

Note: When you delete an Administration MBean, a WebLogic Server instance does not delete the corresponding Configuration MBean until you restart the server instance.

Syntax

```
java [ SSL Arguments ] weblogic.Admin
     [ {-url URL} |
       {-adminurl[protocol://]Admin-Server-listen-address:listen-port}
     ]
     [ User Credentials Arguments ]
   DELETE {-type mbeanType|-mbean objectName}
```

Arguments	Definition

`{-url [protocol://]listen-address:listen-port}`
or
`{-adminurl [protocol://]Admin-Server-listen-address:listen-port}`
	To delete Administration MBeans, use `-url` to specify the Administration Server's listen address and listen port.
	To delete Runtime MBeans or Local Configuration MBeans, use one of the following:
	• `-url` to specify the listen address and listen port of the server instance on which you want to delete MBeans.
	• `-adminurl` to delete instances of a Runtime or Local Configuration MBean type from all server instances in the domain.
	For more information, refer to the `-url` and `-adminurl` entries in Table 1-3 on page 11 and "Protocol Support" on page 1-13.

`-type mbeanType`	Deletes all MBeans of the specified type. For more information, refer to "Specifying MBean Types" on page 1-62.
`-mbean objectName`	Fully qualified object name of an MBean in the `WebLogicObjectName` format. For example: `"domain:Type=type,Name=name"` For more information, refer to the Javadoc for `WebLogicObjectName`.

Example

The following example deletes the `JDBCConnectionPool` Administration MBean named `myPool`:

```
java weblogic.Admin -url AdminHost:7001 -username weblogic
 -password weblogic DELETE -mbean
 MedRec:Name=myPool,Type=JDBCConnectionPool
```

For more information about the environment in which this example runs, refer to "Example Environment" on page 1-14.

If the command succeeds, it prints the following to standard out:

```
Ok
```

The following example deletes the `JDBCConnectionPool` Local Configuration MBean named `myPool` on a server instance named MedRecManagedServer:

```
java weblogic.Admin -url ManagedHost:8001 -username weblogic
 -password weblogic DELETE -mbean
```

```
  MedRec:Location=MedRecManagedServer,Name=myPool,
 Type=JDBCConnectionPoolConfig
```

The following example deletes all `JDBCConnectionPool` Local Configuration MBeans for all server instances in the domain:

```
java weblogic.Admin -adminurl AdminHost:7001 -username weblogic
   -password weblogic DELETE -type JDBCConnectionPoolConfig
```

The following example deletes all `JDBCConnectionPool` Local Configuration MBeans on a server instance named `MedRecManagedServer`:

```
java weblogic.Admin -url ManagedHost:8001 -username weblogic
 -password weblogic DELETE -type JDBCConnectionPoolConfig
```

GET

Displays MBean properties (attributes) and JMX object names (in the `WebLogicObjectName` format).

The output of the command is as follows:

```
{MBeanName object-name {property1 value} {property2 value}. . .}
. . .
```

Note that the properties and values are expressed as name-value pairs, each of which is returned within curly brackets. This format facilitates parsing of the output by a script.

If `-pretty` is specified, each property-value pair is displayed on a new line and curly brackets are not used to separate the pairs:

```
MBeanName: object-name
property1: value
property2: value
.
.
.
MBeanName: object-name
property1: value
abbribute2: value
```

Syntax

```
java [ SSL Arguments ] weblogic.Admin
     [ {-url URL} |
       {-adminurl[protocol://]Admin-Server-listen-address:listen-port}
     ]
     [ User Credentials Arguments ]
   GET [-pretty] {-type mbeanType|-mbean objectName}
   [-property property1] [-property property2]...
```

Argument	Definition

`{-url [protocol://]listen-address:listen-port}`
or
`{-adminurl [protocol://]Admin-Server-listen-address:listen-port}`
	To retrieve Administration MBeans, use `-url` to specify the Administration Server's listen address and listen port.
	To retrieve Runtime MBeans or Local Configuration MBeans, use one of the following:
	• `-url` to specify the listen address and listen port of the server instance on which you want to retrieve MBeans.
	• `-adminurl` to retrieve instances of a Runtime or Local Configuration MBean type from all server instances in the domain.
	For more information, refer to the `-url` and `-adminurl` entries in Table 1-3 on page 11 and "Protocol Support" on page 1-13.

`-type mbeanType`	Returns information for all MBeans of the specified type. For more information, refer to "Specifying MBean Types" on page 1-62.
`-mbean objectName`	Fully qualified object name of an MBean in the `WebLogicObjectName` format: `"domain:Type=type,Location=location,Name=name"` For more information, refer to the Javadoc for `WebLogicObjectName`.
`-pretty`	Places property-value pairs on separate lines.
`-property property`	The name of the MBean property (attribute) or properties to be listed. **Note:** If property is not specified using this argument, all properties are displayed.

Example

The following example displays all properties of the `JDBCConnectionPool` Administration MBean for a connection pool named `MedRecPool`. Note that the command must connect to the Administration Server to retrieve information from an Administration MBean:

```
java weblogic.Admin -url AdminHost:7001 -username weblogic
 -password weblogic GET -pretty -mbean
 MedRec:Name=MedRecPool,Type=JDBCConnectionPool
```

For more information about the environment in which this example runs, refer to "Example Environment" on page 1-14.

If the command succeeds, it returns output similar to the following truncated example:

```
---------------------------
MBeanName: "MedRec:Name=MedRecPool,Type=JDBCConnectionPool"
        ACLName:
        CachingDisabled: true
        CapacityIncrement: 1
        ConnLeakProfilingEnabled: false
        ConnectionCreationRetryFrequencySeconds: 0
        ConnectionReserveTimeoutSeconds: 10
...
```

The following example displays all instances of all `JDBCConnectionPoolRuntime` MBeans for all servers in the domain.

```
java weblogic.Admin -adminurl AdminHost:7001 -username weblogic
 -password weblogic GET -pretty -type JDBCConnectionPoolRuntime
```

The following example displays all instances of all `JDBCConnectionPoolRuntime` MBeans that have been deployed on the server instance that listens on `ManagedHost:8001`:

```
java weblogic.Admin -url ManagedHost:8001 -username weblogic
 -password weblogic GET -pretty -type JDBCConnectionPoolRuntime
```

INVOKE

Invokes a management operation for one or more MBeans. For WebLogic Server MBeans, you usually use this command to invoke operations other than the `get`*`Attribute`* and `set`*`Attribute`* that most WebLogic Server MBeans provide.

Syntax

```
java [ SSL Arguments ] weblogic.Admin
     [ {-url URL} |
       {-adminurl[protocol://]Admin-Server-listen-address:listen-port}
     ]
     [ User Credentials Arguments ]
   INVOKE {-type mbeanType|-mbean objectName} -method
   methodname [argument . . .]
```

Arguments	Definition

{-url [*protocol*://]*listen-address*:*listen-port*}
or
{-adminurl [*protocol*://]*Admin-Server-listen-address*:*listen-port*}
	To invoke operations for Administration MBeans, use -url to specify the Administration Server's listen address and listen port.
	To invoke operations for Runtime MBeans, use one of the following:
	• -url to specify the listen address and listen port of the server instance on which you want to invoke Runtime MBean operations.
	• -adminurl to invoke operations for all instances of a Runtime MBean on all server instances in the domain.
	We recommend that you do not invoke operations for Local Configuration MBeans. Instead, invoke the operation on the corresponding Administration MBean.

-type *mbeanType*	Invokes the operation on all MBeans of a specific type. For more information, refer to "Specifying MBean Types" on page 1-62.
-mbean *objectName*	Fully qualified object name of an MBean, in the WebLogicObjectName format: "*domain*:Type=*type*,Location=*location*,Name=*name*" For more information refer to the Javadoc for WebLogicObjectName.
-method *methodname*	Name of the method to be invoked.
argument	Arguments to be passed to the method call. When the argument is a String array, the arguments must be passed in the following format: "*String1*;*String2*;. . . "

Example

The following example enables a JDBC connection pool by invoking the `enable` method of the `JDBCConnectionPoolRuntime` MBean:

```
java weblogic.Admin -url AdminHost:7001 -username weblogic
 -password weblogic INVOKE
 -mbean MedRec:Location=MedRecServer,Name=myPool,
  ServerRuntime=MedRec,Type=JDBCConnectionPoolRuntime
 -method enable
```

If the command succeeds, it returns the following:

```
{MBeanName="MedRec:Location=MedRecServer,Name=MedRecPool,ServerRuntime=Med
RecServer,Type=JDBCConnectionPoolRuntime"}

Ok
```

For more information about the environment in which this example runs, refer to "Example Environment" on page 1-14.

The following example enables all JDBC connection pools in the domain by invoking the `enable` method of all the `JDBCConnectionPoolRuntime` MBeans:

```
java weblogic.Admin -adminurl AdminHost:7001 -username weblogic
 -password weblogic INVOKE
 -type JDBCConnectionPoolRuntime -method enable
```

QUERY

Searches for WebLogic Server MBeans whose `WebLogicObjectName` matches a pattern that you specify.

All MBeans that are created from a WebLogic Server MBean type are registered in the MBean Server under a name that conforms to the `weblogic.management.WebLogicObjectName` conventions. You must know an MBean's `WebLogicObjectName` if you want to use `weblogic.Admin` commands to retrieve or modify specific MBean instances. For more information, refer to "WebLogic Server Management Namespace" in the *Programming WebLogic Management Services with JMX* guide.

The output of the command is as follows:

```
{MBeanName object-name {property1 value} {property2 value}. . .}
. . .
```

Note that the properties and values are expressed as name-value pairs, each of which is returned within curly brackets. This format facilitates parsing of the output by a script.

If `-pretty` is specified, each property-value pair is displayed on a new line and curly brackets are not used to separate the pairs:

```
MBeanName: object-name
property1: value
property2: value
.
.
.
MBeanName: object-name
property1: value
abbribute2: value
```

Syntax

```
java [ SSL Arguments ] weblogic.Admin
     [ {-url URL} |
       {-adminurl[protocol://]Admin-Server-listen-address:listen-port}
     ]
     [ User Credentials Arguments ]
   QUERY -pretty -pattern object-name-pattern
```

Argument	Definition

`{-url [protocol://]listen-address:listen-port}`
or
`{-adminurl [protocol://]Admin-Server-listen-address:listen-port}`
	To search for Administration MBean object names, use `-url` to specify the Administration Server's listen address and listen port.
	To search for the object names of Local Configuration or Runtime MBeans, use one of the following:
	• `-url` to specify the listen address and listen port of the server instance on which you want to search.
	• `-adminurl` to search on all server instances in the domain.
	For more information, refer to the `-url` and `-adminurl` entries in Table 1-3 on page 11 and "Protocol Support" on page 1-13.

Argument	Definition
-pretty	Places property-value pairs on separate lines.
-pattern	A partial WebLogicObjectName for which the QUERY command searches. The value must conform to the following pattern: *domain-name*:*property-list* For the *domain-name* portion of the pattern, you can use the * character, which matches any character sequence. Because the server instance that you specify with the -url or -adminurl argument can access only the MBeans that belong to its domain, the * character is sufficient. For example, if you use -url to specify a server in the MedRec domain, QUERY can only return MBeans that are in the MedRec domain. It cannot search for MBeans in a domain named mydomain. For the *property-list* portion of the pattern, specify one or more components (property-value pairs) of a WebLogicObjectName. For a list of all WebLogicObjectName property-value pairs, refer to "WebLogic Server Management Namespace" in the *Programming WebLogic Management Services with JMX* guide. (For example, all WebLogicObjectNames include Name=*value* and Type=*value* property-value pairs.) You can specify these property-value pairs in any order. Within a given naming property-value pair, there is no pattern matching. Only complete property-value pairs are used in pattern matching. However, you can use the * wildcard character in the place of one or more property-value pairs. For example, Name=Med* is not valid, but Name=MedRecServer,* is valid. If you provide at least one property-value pair in the *property-list*, you can locate the wildcard anywhere in the given pattern, provided that the *property-list* is still a comma-separated list.

Example

The following example searches for all JDBCConnectionPoolRuntime MBeans that are on a server instance that listens at ManagedHost:8001:

```
java weblogic.Admin -url ManagedHost:8001 -username weblogic
 -password weblogic QUERY
 -pattern *:Type=JDBCConnectionPoolRuntime,*
```

If the command succeeds, it returns the following:

```
Ok
```

For more information about the environment in which this example runs, refer to "Example Environment" on page 1-14.

The following example searches for all instances of MedRecPool MBeans on all servers in the current domain. It uses -adminurl, which instructs the Administration Server to query the Administration MBeanHome interface (This interface has access to all MBeans in the domain):

```
java weblogic.Admin -adminurl AdminHost:7001 -username weblogic
 -password weblogic QUERY -pattern *:Name=MedRecPool,*
```

If the command succeeds, it returns an instance of the JDBCConnectionPool Administration MBean that is named MedRecPool, along with all corresponding Local Configuration and Runtime MBeans.

SET

Sets the specified property (attribute) values for a configuration MBean. This command cannot be used for Runtime MBeans.

If the command is successful, it returns OK.

When you use this command for an Administration MBean, the new values are saved to the config.xml file.

We recommend that you do not use this command to set values on a Local Configuration MBean. If you use this command for a Local Configuration MBean, the new values are not saved to the config.xml file. Depending on the attribute that you set, the subsystem that uses the MBean might not be able to modify its operation per the new value. In addition, some subsystems require that their Local Configuration MBeans be replicated throughout a domain. If you modify the value for a Local Configuration MBean on one server, the new value will not be replicated throughout the domain and the subsystem might not operate correctly.

Syntax

```
java [ SSL Arguments ] weblogic.Admin
     [-url URL]
     [ User Credentials Arguments ]
   SET {-type mbeanType|-mbean objectName}
   -property property1 property1_value
   [-property property2 property2_value] . . .
```

Argument	Definition
-url [*protocol*://]*listen-address*:*listen-port*	Specifies the listen address and listen port of the Administration Server. Only the Administration Server can access Administration MBeans. If you specify a secure listen port, you must also specify a secure protocol. If you do not specify a value, the command assumes t3://localhost:7001. For more information, refer to the -url entry in Table 1-3 on page 11 and "Protocol Support" on page 1-13. Although the SET command supports the -adminurl, we recommend that you do not use it to set values of Local Configuration MBeans.
-type *mbeanType*	Sets the properties for all MBeans of a specific type. For more information, refer to "Specifying MBean Types" on page 1-62.
-mbean *objectName*	Fully qualified object name of an MBean in the WebLogicObjectName format. For example: "*domain*:Type=*type*,Name=*name*" For more information, refer to the Javadoc for WebLogicObjectName.

Argument	Definition
-property *property*	The name of the property to be set.
property _value	The value to be set. • Some properties require you to specify the name of a WebLogic Server MBean. In this case, specify the fully qualified object name of an MBean in the WebLogicObjectName format. For example: “*domain*:Type=*type*,Name=*name*” For more information, refer to the Javadoc for WebLogicObjectName. • When the property value is an MBean array, separate each MBean object name by a semicolon and surround the entire property value list with quotes: “*domain*:Name=*name*,Type=*type*;*domain*:Name=*name*,Type=*type*” • When the property value is a String array, separate each string by a semicolon and surround the entire property value list with quotes: “*String1*;*String2*;. . . ” • When the property value is a String or String array, you can set the value to null by using either of the following: -property *property-name* "" -property *property-name* For example, both -property ListenAddress "" and -property ListenAddress set the listen address to null. • If the property value contains spaces, surround the value with quotes: "-Da=1 -Db=3" For example: SET -type ServerStart -property Arguments "-Da=1 -Db=3" • When setting the properties for a JDBC Connection Pool, you must pass the arguments in the following format: “user:*username*;password:*password*;server:*servername*”

Example

The following example sets to 64 the StdoutSeverityLevel property of the local configuration instance of the ServerMBean for a server named MedRecManagedServer:

```
java weblogic.Admin -url http://ManagedHost:8001
  -username weblogic -password weblogic
  SET -mbean
  MedRec:Location=MedRecManagedServer,Name=MedRecManagedServer,
  Type=ServerConfig
 -property StdoutSeverityLevel 64
```

For more information about the environment in which this example runs, refer to "Example Environment" on page 1-14.

If the command succeeds, the server instance writes a log message similar to the following:

```
<Sep 16, 2002 12:11:27 PM EDT> <Info> <Logging> <000000> <Log messages of
every severity will be displayed in the shell console.>
```

The command prints `Ok` to standard out.

The following example sets to `64` the `StdoutSeverityLevel` property for all administration instances of `ServerMBean` in the current domain:

```
java weblogic.Admin -url http://AdminHost:7001
  -username weblogic -password weblogic
  SET -type Server -property StdoutSeverityLevel 64
```

Running Commands in Batch Mode

By default, each `weblogic.Admin` command that you invoke starts a JVM, acts on a server instance, and then shuts down the JVM. To improve performance for issuing several `weblogic.Admin` commands in an uninterrupted sequence, you can use the `BATCHUPDATE` command to run multiple commands in batch mode. The `BATCHUPDATE` command starts a JVM, runs a list of commands, and then shuts down the JVM.

For example, if a domain contains multiple server instances, you can create a file that returns the listen ports of all Managed Servers in a domain. Then you specify this file as an argument in `weblogic.Admin BATCHUPDATE` command.

BATCHUPDATE

Runs a sequence of `weblogic.Admin` commands. All output from commands that `BATCHUPDATE` runs is printed to standard out.

Using this command provides better performance than issuing a series of individual `weblogic.Admin` commands. For more information, refer to the previous section, "Running Commands in Batch Mode" on page 1-78.

Syntax

```
java [ SSL Arguments ] weblogic.Admin
     [ {-url URL} |
       {-adminurl[protocol://]Admin-Server-listen-address:listen-port}
     ]
     [ User Credentials Arguments ]
    BATCHUPDATE -batchFile fileLocation
   [-continueOnError] [-batchCmdVerbose]
```

Argument	Definition

`{-url [protocol://]listen-address:listen-port}`
or
`{-adminurl [protocol://]Admin-Server-listen-address:listen-port}`
	If the batch file contains commands that access Administration MBeans, use `-url` to specify the Administration Server's listen address and listen port.
	If the batch file contains commands that access Local Configuration or Runtime MBeans, use one of the following:
	• `-url` to specify the listen address and listen port of the server instance on which you want to access MBeans.
	• `-adminurl` to access all Local Configuration or Runtime MBeans in the domain.
	If you specify a secure listen port, you must also specify a secure protocol.
	If you do not specify a value, the command assumes `t3://localhost:7001`.
	For more information, refer to the `-url` and `-adminurl` entries in Table 1-3 on page 11 and "Protocol Support" on page 1-13.

`-batchfile fileLocation`	The name of a text file that contains a list of `weblogic.Admin` commands. If you use a relative pathname, the root context is the directory from which you issue the `weblogic.Admin BATCHUPDATE` command. The file must contain one or more commands, formatted as follows: *`COMMAND-NAME arguments`* Place each command on a separate line. Within the batch file, the `BATCHUPDATE` command ignores any line that begins with a # character. **Note:** Quoted MBean names are allowed in the batch file.

Argument	Definition
`-continueOnError`	If one of the commands fails or emits errors, `weblogic.Admin` ignores the error and continues to the next command. By default, `weblogic.Admin` stops processing commands as soon as it encounters an error.
`-batchCmdVerbose`	Causes `BATCHUPDATE` to indicate which command it is currently invoking. As it invokes a command, `BATCHUPDATE` prints the following to standard out: `Executing command:` *`command-from-batchfile`*

Example

This example uses the `BATCHUPDATE` command to return the listen ports for a collection of server instances in a domain. A file named `commands.txt` contains the following lines:

```
get -mbean MedRec:Name=MedRecServer,Type=Server -property ListenPort
get -mbean MedRec:Name=MedRecManagedServer,Type=Server -property ListenPort
```

The following command invokes the commands in `commands.txt`:

```
java weblogic.Admin -url AdminHost:7001 -username weblogic -password
weblogic BATCHUPDATE -batchFile c:\commands.txt -continueOnError
-batchCmdVerbose
```

If the command succeeds it outputs the following to standard out:

```
Executing command: get -mbean MedRec:Name=MedRecServer,Type=Server -property
ListenPort
{MBeanName="MedRec:Name=MedRecServer,Type=Server"{ListenPort=7001}}
Executing command: get -mbean MedRec:Name=MedRecManagedServer,Type=Server
-property ListenPort
{MBeanName="MedRec:Name=MedRecManagedServer,Type=Server"{ListenPort=7021}}
```

For information about the environment in which this example runs, refer to "Example Environment" on page 1-14.

Commands for Working with Clusters

Table 1-8 is an overview of the commands for working with clusters. Subsequent sections describe command syntax and arguments, and provide an example for each command.

Table 1-8 MBean Management Command Overview

Command	Description
CLUSTERSTATE	Returns the number and state of servers in a cluster. See "CLUSTERSTATE" on page 1-81.
MIGRATE	Migrates a JMS service or a JTA service from one server instance to another within a cluster. See "MIGRATE" on page 1-82.
STARTCLUSTER	Starts all servers in a cluster See "STARTCLUSTER" on page 1-84.
STOPCLUSTER	Forces all servers in a cluster to shut down. See "STOPCLUSTER" on page 1-86.
VALIDATECLUSTERCONFIG	Parses the domain's configuration file and reports any discrepancies in all cluster-related elements. See "VALIDATECLUSTERCONFIG" on page 1-87.

CLUSTERSTATE

Returns the number and state of servers in a cluster.

Syntax

```
java [ SSL Arguments ] weblogic.Admin
     [-url URL]
     [ User Credentials Arguments ]
   CLUSTERSTATE -clusterName clusterName
```

Argument	Definition
`{-url [`*`protocol`*`://]`*`listen-address`*`:`*`listen-port`*`}`	Specify the listen address and listen port of any server instance that is currently active and that belongs to the cluster. If you specify a secure listen port, you must also specify a secure protocol. If you do not specify a value, the command assumes `t3://localhost:7001`. For more information, refer to the `-url` entry in Table 1-3 on page 11 and "Protocol Support" on page 1-13.
`-clusterName` *`clusterName`*	The name of the cluster as specified in the domain's configuration file (`config.xml`).

Example

The following example returns information about a cluster:

```
java weblogic.Admin -url AdminHost:7001
  -username weblogic -password weblogic
  CLUSTERSTATE -clustername MedRecCluster
```

For more information about the environment in which this example runs, refer to "Example Environment" on page 1-14.

If the command succeeds, it returns output similar to the following:

```
There are 3 server(s) in cluster: MedRecCluster

The alive servers and their respective states are listed below:
MedRecManagedServer2---RUNNING
MedRecManagedServer3---RUNNING

The other server(s) in the cluster that are not active are:
MedRecManagedServer1
```

MIGRATE

Migrates a JMS service or a JTA Transaction Recovery service to a targeted server within a server cluster.

For more information about migrating services, refer to "Migration for Pinned Services" in the *Using WebLogic Clusters* guide.

Syntax

To migrate JMS resources:

```
java [ SSL Arguments ] weblogic.Admin
     [-url URL]
     [ User Credentials Arguments ]
  MIGRATE -migratabletarget "serverName (migratable)"
  -destination serverName [-sourcedown] [-destinationdown]
```

To migrate JTA resources:

```
java [ SSL Arguments ] weblogic.Admin
     [-url URL]
     [ User Credentials Arguments ]
  MIGRATE -jta -migratabletarget serverName
  -destination serverName [-sourcedown] [-destinationdown]
```

Argument	Definition
`{-url [protocol://]listen-address:listen-port}`	Specify the listen address and listen port of the Administration Server. If you specify a secure listen port, you must also specify a secure protocol. If you do not specify a value, the command assumes `t3://localhost:7001`. For more information, refer to the `-url` entry in Table 1-3 on page 11 and "Protocol Support" on page 1-13.
`-jta`	Specifies that the migration is a migration of JTA services. If you do not specify this argument, the `MIGRATE` command migrates JMS services.
`-migratabletarget`	Names the server from which the service will migrate. The syntax for the server name varies depending on the type of service you are migrating: • For JMS, specify `"servername (migratable)"` For example, `"myserver (migratable)"` • For JTA, specify `servername` For example, `myserver`

Argument	Definition
`-destination`	Names the server to which the service will migrate.
`-sourcedown`	Specifies that the source server is down. This switch should be used very carefully. If the source server is not in fact down, but only unavailable because of network problems, the service will be activated on the destination server without being removed from the source server, resulting in two simultaneous running versions of the same service, *which could cause corruption of the transaction log or of JMS messages*.
`-destinationdown`	Specifies that the destination server is down. When migrating a JMS service to a non-running server instance, the server instance will activate the JMS service upon the next startup. When migrating the JTA Transaction Recovery Service to a non-running server instance, the target server will assume recovery services when it is started.

Examples

In the following example, a JMS service is migrated from myserver2 to myserver3.

```
java weblogic.Admin -url AdminHost:7001 -username weblogic
   -password weblogic
   MIGRATE -migratabletarget "myserver2 (migratable)"
  -destination myserver3
```

In the following example, a JTA Transaction Recovery service is migrated from myserver2 to myserver3.

```
java weblogic.Admin -url AdminHost:7001 -username weblogic
   -password weblogic
    MIGRATE -jta -migratabletarget myserver2
   -destination myserver3 -sourcedown
```

STARTCLUSTER

Starts all of the servers that are in a cluster have been configured to use a Node Manager.

This command requires the following environment:

- The domain’s Administration Server must be running.
- The Node Manager must be running on the Managed Server’s host machine.

- The Managed Server must be configured to communicate with a Node Manager. For more information, refer to “Configuring a Machine” in the Administration Console Online Help.

The Startup Mode field in the Administration Console determines whether a Managed Server starts in the `RUNNING` state or `STANDBY` state. See "Server Life Cycle" in *Configuring and Managing WebLogic Server*.

Syntax

```
java [ SSL Arguments ] weblogic.Admin
     [-url URL]
     [ User Credentials Arguments ]
   STARTCLUSTER -clusterName clusterName
```

Argument	Definition
`{-url [protocol://]listen-address:listen-port}`	Specify the listen address and listen port of the Administration Server. If you specify a secure listen port, you must also specify a secure protocol. If you do not specify a value, the command assumes `t3://localhost:7001`. For more information, refer to the `-url` entry in Table 1-3 on page 11 and “Protocol Support” on page 1-13.
`-clusterName clusterName`	The name of the cluster as specified in the domain’s configuration file (`config.xml`).

Example

The following example starts a cluster:

```
java weblogic.Admin -url AdminHost:7001
  -username weblogic -password weblogic
  STARTCLUSTER -clustername MedRecCluster
```

For more information about the environment in which this example runs, refer to “Example Environment” on page 1-14.

If the command succeeds, it returns output similar to the following:

```
Starting servers in cluster MedRecCluster: MedRecMS2,MedRecMS1
All servers in the cluster "MedRecCluster" started successfully.
```

STOPCLUSTER

Forces all servers in a cluster to shut down without waiting for active sessions to complete.

To verify that the command succeeds for a given server instance, refer to the server's local message log and look for the following message:
`<BEA-000238> <Shutdown has completed.>`

Review the message time stamp to verify that it was generated by the server session for which you issued the stop command.

For more information about forced shutdowns, refer to "Forced Shutdown" in the *Configuring and Managing WebLogic Server* guide.

Syntax

```
java [ SSL Arguments ] weblogic.Admin
     [-url URL]
     [ User Credentials Arguments ]
   STOPCLUSTER -clusterName clusterName
```

Argument	Definition
`{-url [protocol://]listen-address:listen-port}`	Specify the listen address and listen port of the Administration Server. If you specify a secure listen port, you must also specify a secure protocol. If you do not specify a value, the command assumes `t3://localhost:7001`. For more information, refer to the `-url` entry in Table 1-3 on page 11 and "Protocol Support" on page 1-13.
`-clusterName clusterName`	The name of the cluster as specified in the domain's configuration file (`config.xml`).

Example

The following example stops a cluster:

```
java weblogic.Admin -url AdminHost:7001
  -username weblogic -password weblogic
  STOPCLUSTER -clustername MedRecCluster
```

For more information about the environment in which this example runs, refer to "Example Environment" on page 1-14.

If the command succeeds, it returns output similar to the following:

```
Shutting down servers in cluster MedRecCluster: MedRecMS2,MedRecMS1
All servers in the cluster "MedRecCluster" were issued the shutdown request.
Look in the server logs to verify the success or failure of the shutdown
request.
```

VALIDATECLUSTERCONFIG

Parses the domain's configuration file and reports any errors in the configuration of cluster-related elements.

You can run this command only on a WebLogic Server host that can access the domain's configuration file through the host's file system.

Syntax

```
java [ SSL Arguments ] weblogic.Admin
     [-url URL]
     [ User Credentials Arguments ]
   VALIDATECLUSTERCONFIG
   -configPath pathname
```

Argument	Definition
`{-url [protocol://]listen-address:listen-port}`	Specify the listen address and listen port of any active server in the domain, regardless of whether it belongs to a cluster. If you specify a secure listen port, you must also specify a secure protocol. If you do not specify a value, the command assumes `t3://localhost:7001`. For more information, refer to the `-url` entry in Table 1-3 on page 11 and "Protocol Support" on page 1-13.
`-configPath pathname`	The path and file name of the domain's configuration file. A relative pathname is resolved to the directory in which you issue the `VALIDATECLUSTERCONFIG` command.

Example

The following example validates the cluster-related configuration elements for the MedRec domain. In this example, the command is issued from the *WL_HOME* directory:

```
java weblogic.Admin -url AdminHost:7001
  -username weblogic -password weblogic
  VALIDATECLUSTERCONFIG -configpath
  samples\domains\medrec\config.xml
```

For more information about the environment in which this example runs, refer to "Example Environment" on page 1-14.

If the cluster configuration contains errors, the command returns a message that describes the error. For example:

```
ERROR:Cluster name:MyCluster has an INVALID Multicast address:null Please
pick an address between (224.0.0.1 and 255.255.255.255)
```

If the cluster configuration is free of errors, the command returns nothing.

Command for Purging Tasks

When the `weblogic.Deployer` tool is used to deploy or undeploy applications to WebLogic Server, `DeploymentTaskRuntimeMBeans` are created on the server side to handle each of these deployment tasks. These `TaskRuntimeMBeans`, created to execute deployment requests are not automatically purged after the completion of the deployment tasks. Instead, they are deleted only when they are manually purged from WebLogic Server Administration Console.

PURGETASKS

You can use the `PURGETASKS` command to purge `DeploymentTaskRuntimeMBeans` on the server side. Purging these tasks avoids any Deployer related memory leaks on the server-side.

Syntax

```
java [ SSL Arguments ] weblogic.Admin
     [ Connection Arguments ]
     [ User Credentials Arguments ]
PURGETASKS
```

Examples

The following example connects through a non-secured listen port.

```
java weblogic.Admin -url t3://localhost:7001 -username weblogic -password
weblogic PURGETASKS
```

The following example connects through an SSL listen port on a server that is using the demonstration SSL keys and certificates.

```
java -Dweblogic.security.TrustKeyStore=DemoTrust weblogic.Admin -url
t3s://localhost:7002 -username weblogic -password weblogic PURGETASKS
```

Using Command-Line Utilities to Configure a WebLogic Server Domain

WebLogic Server provides several command-line utilities that you can use to automate the creation of domains, servers, and resources. The following sections describe combining shell scripts with the `weblogic.Server`, `weblogic.Admin`, and `weblogic.Deployer` commands to automate typical configuration tasks:

- "Using Command-Line Utilities to Clone the MedRec Domain: Main Steps" on page 2-2
- "Using weblogic.Admin Commands to Manage Users and Groups" on page 2-20
- "Using weblogic.Admin Commands to Target Resources" on page 2-29
- "Using weblogic.Admin Commands to Configure the WebLogic SNMP Agent" on page 2-35
- "Using weblogic.Admin Commands to Create a Simple Cluster" on page 2-38

Alternatively, you use one of the following techniques to automate the configuration of a WebLogic Server domain based on your familiarity with the WebLogic Server Template Builder or Ant:

- Use the Template Builder to save an application, server configuration, and startup script as a configuration template. Then use the Configuration Wizard to create domains based on the configuration template. The Configuration Wizard includes a silent install option, which enables you to duplicate a domain on multiple machines without user interaction.

  See "Creating Configuration Templates Using the Template Builder" in *Creating WebLogic Configurations Using the Configuration Wizard* and "Starting in Silent Mode" in *Configuring WebLogic Platform*.

Note: Although the documentation for the Template Builder and Configuration Wizard utilities is part of the WebLogic Platform documentation set, the utilities are installed with WebLogic Server and available for use even if you do not install WebLogic Platform.

- Use the WebLogic Server Ant tasks. For almost all configuration needs, the Ant tasks and the `weblogic.Server`, `weblogic.Admin`, and `weblogic.Deployer` commands are functionally equivalent.

  See "Using Ant Tasks to Configure a WebLogic Server Domain" on page 5-1.

Using Command-Line Utilities to Clone the MedRec Domain: Main Steps

The section "Sample Korn Shell Script" on page 2-13 provides a sample script for cloning and slightly modifying the MedRec domain.

To create and configure a domain such as the MedRec sample domain, use a shell script to:

1. Set the required environment variables.

   See "Setting Up the Environment" on page 2-3.

2. Create a skeletal domain with the `java weblogic.Server` command:

   See "Creating a Domain" on page 2-3.

3. After the skeletal domain's Administration Server has completed its startup cycle, configure resources for the domain by invoking a series of `weblogic.Admin` commands that instantiate Administration MBeans. See:

   – "Creating JDBC Resources" on page 2-4

   – "Creating JMS Resources" on page 2-6

   – "Creating Mail Resources" on page 2-9

   – "Creating and Configuring Managed Servers" on page 2-9

   – "Configuring Machines and Node Manager Properties" on page 2-10

   For more information about MBeans in WebLogic Server, see "Overview of WebLogic JMX Services" in *Programming WebLogic Management Services with JMX*.

4. Invoke multiple `weblogic.Deployer` commands to deploy J2EE modules such as EJBs and Enterprise applications.

See “Deploying Applications” on page 2-12.

Setting Up the Environment

All WebLogic Server commands require an SDK to be specified in the environment’s `PATH` variable and a set of WebLogic Server classes to be specified in the `CLASSPATH` variable.

Use the following script to add an SDK to `PATH` variable and the WebLogic Server classes to the `CLASSPATH` variable:

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

If you want to use JDBC resources to connect to a database, modify the environment as the database vendor requires. Usually this entails adding driver classes to the `CLASSPATH` variable and vendor-specific directories to the `PATH` variable. To set the environment that the sample Pointbase database requires as well as add an SDK to `PATH` variable and the WebLogic Server classes to the `CLASSPATH` variable, invoke the following script:

`WL_HOME\samples\domains\medrec\setMedRecEnv.cmd` (on Windows)
`WL_HOME/samples/domains/medrec/setMedRecEnv.sh` (on UNIX)

Creating a Domain

To create a domain named mymedrec with an Administration Server named myMedRecServer that listens on port 8001:

1. Create an empty directory named `mymedrec`.

2. Change to the empty directory and enter the following command:

```
java -Dweblogic.management.username=weblogic
    -Dweblogic.management.password=weblogic
    -Dweblogic.Domain=mymedrec
    -Dweblogic.Name=myMedRecServer
    -Dweblogic.ListenPort=8001
    -Dweblogic.management.GenerateDefaultConfig=true
     weblogic.Server
```

   When you invoke the `weblogic.Server` class in a directory that does not contain a `config.xml` file, WebLogic Server creates and starts a domain with an Administration Server. See “Default Behavior” on page 4-3.

   Notes:

– The value of the `-Dweblogic.Domain` option must match the name of the current directory.

– The command specifies a listen port of 8001 because the sample MedRec domain that WebLogic Server installs listens on the default port 7001. If the sample MedRec domain is running, the 7001 listen port cannot be used by another server instance.

– The `-Dweblogic.management.GenerateDefaultConfig=true` prevents the `weblogic.Server` class from prompting you for confirmations.

Creating JDBC Resources

The commands in Listing 2-1 create JDBC resources in the mymedrec domain.

Listing 2-1 Creating JDBC Resources

```
# Create a JDBC Connection Pool for Applications
CREATE_POOL myMedRecPoolXA
driver="com.pointbase.xa.xaDataSource",
url=jdbc:pointbase:server://localhost:9093/demo,
props=user=medrec;password=medrec;
DatabaseName=jdbc:pointbase:server://localhost:9093/demo,maxCapacity=10

# Create a Transactional Data Source
CREATE -mbean mymedrec:Name=MedRecTxDataSource,Type=JDBCTxDataSource

# Configure the Transactional Data Source
SET -mbean mymedrec:Name=MedRecTxDataSource,Type=JDBCTxDataSource
-property JNDIName "MedRecTxDataSource"

SET -mbean mymedrec:Name=MedRecTxDataSource,Type=JDBCTxDataSource
-property PoolName "myMedRecPoolXA"

# Create another JDBC Connection Pool for the JMS JDBC Store
CREATE_POOL myMedRecPool driver="com.pointbase.jdbc.jdbcUniversalDriver",
url=jdbc:pointbase:server://localhost:9093/demo,
props=user=medrec;password=medrec,
DatabaseName=jdbc:pointbase:server://localhost:9093/demo,maxCapacity=10
```

When invoked by the `weblogic.Admin BATCHUPDATE` command, the commands in Listing 2-1 do the following (see "BATCHUPDATE" on page 1-78):

1. Create a JDBC connection pool and configure it to connect to the MedRec sample PointBase database, which WebLogic Server installs.

   The database listens on port 9093 and is named demo.

   The `CREATE_POOL` command creates an MBean of type `JDBCConnectionPool` whose object name is `mymedrec:Name=myMedRecPoolXA,Type=JDBCConnectionPool`.

   For more information, see:

   – "CREATE_POOL" on page 1-53

   – "JDBC Configuration Namespace" in *Programming WebLogic Management Services with JMX.*

2. Create a JDBC transactional data source by instantiating a `JDBCTxDataSourceMBean` Administration MBean.

   The `CREATE -mbean` command creates an MBean and specifies the MBean's `WebLogicObjectName`. In this object name:

   – `mymedrec`: specifies the name of the WebLogic Server domain.

   – `Name=MedRecTxDataSource` provides a unique name for the JDBC transactional data source you want to create.

   – `Type=JDBCTxDataSource` specifies the type of MBean to create.

3. Configure the transactional data source by setting attributes on the `JDBCTxDataSourceMBean` that you instantiated.

   To see all attributes and legal values of the `JDBCTxDataSourceMBean`, refer to the Javadoc.

4. Create an additional JDBC connection pool that uses the universal driver instead of the XA driver. This connection pool will be used by the JMS resources, which do not support the XA driver.

Using the Sample Commands

To create JDBC resources in the mymedrec domain:

1. Copy the commands in Listing 2-1 and paste them into an empty text file.

2. Open a command prompt (shell) and invoke `*WL_HOME*\common\bin\startPointBase.sh`.

   This script starts a PointBase database that the mymedrec domain uses.

3. If you haven't already started the Administration Server for the mymedrec domain:

   a. Change to the `mymedrec` directory that you created in "Creating a Domain" on page 2-3.

   b. Invoke one of the following scripts:

      WL_HOME`\samples\domains\medrec\setMedRecEnv.cmd` (on Windows)
      WL_HOME`/samples/domains/medrec/setMedRecEnv.sh` (on UNIX)

   c. Enter the following command:

```
java -Dweblogic.management.username=weblogic
    -Dweblogic.management.password=weblogic
    -Dweblogic.Name=myMedRecServer
    -Dweblogic.ListenPort=8001
     weblogic.Server
```

4. Open another command prompt (shell) and do the following:

   a. Invoke *WL_HOME*`\samples\domains\medrec\setMedRecEnv.cmd` or `setMedRecEnv.sh`.

   b. Invoke the following command:

```
java weblogic.Admin -url localhost:7001 -username weblogic
-password weblogic BATCHUPDATE -batchFile c:\myfile -batchCmdVerbose
```

   where `c:\myfile` is the name of a text file that contains the commands in Listing 2-1.

   See "BATCHUPDATE" on page 1-78.

Creating JMS Resources

The commands in Listing 2-2 create JMS resources in the mymedrec domain.

Listing 2-2 Creating JMS Resources

```
# Creating a JMS Connection Factory
CREATE -mbean mymedrec:Name=MedRecQueueFactory,Type=JMSConnectionFactory

# Configuring the JMS Connection Factory
SET -mbean mymedrec:Name=MedRecQueueFactory,Type=JMSConnectionFactory
-property JNDIName "jms/MedRecQueueConnectionFactory"

SET -mbean mymedrec:Name=MedRecQueueFactory,Type=JMSConnectionFactory
-property XAServerEnabled "true"
```

```
#
# Creating and Configuring a JMS JDBC Store
CREATE -mbean mymedrec:Name=MedRecJDBCStore,Type=JMSJDBCStore

SET -mbean mymedrec:Name=MedRecJDBCStore,Type=JMSJDBCStore
-property ConnectionPool "mymedrec:Name=myMedRecPool,Type=JDBCConnectionPool"

SET -mbean mymedrec:Name=MedRecJDBCStore,Type=JMSJDBCStore
-property PrefixName "MedRec"

# Creating and Configuring a JMS Server
CREATE -mbean mymedrec:Name=MedRecJMSServer,Type=JMSServer

SET -mbean mymedrec:Name=MedRecJMSServer,Type=JMSServer
-property Store "mymedrec:Name=MedRecJDBCStore,Type=JMSJDBCStore"

# Creating and Configuring a Queue
CREATE -mbean
mymedrec:JMSServer=MedRecJMSServer,Name=RegistrationQueue,Type=JMSQueue

SET -mbean
mymedrec:JMSServer=MedRecJMSServer,Name=RegistrationQueue,Type=JMSQueue
-property JNDIName "jms/REGISTRATION_MDB_QUEUE"

# Creating and Configuring an Additional Queue
CREATE -mbean mymedrec:JMSServer=MedRecJMSServer,Name=MailQueue,Type=JMSQueue

SET -mbean mymedrec:JMSServer=MedRecJMSServer,Name=MailQueue,Type=JMSQueue
-property JNDIName "jms/MAIL_MDB_QUEUE"

# Creating and Configuring an Additional Queue
CREATE -mbean mymedrec:JMSServer=MedRecJMSServer,Name=XMLQueue,Type=JMSQueue

SET -mbean mymedrec:JMSServer=MedRecJMSServer,Name=XMLQueue,Type=JMSQueue
-property JNDIName "jms/XML_UPLOAD_MDB_QUEUE"
```

When invoked by the `weblogic.Admin BATCHUPDATE` command, the commands do the following (see “BATCHUPDATE” on page 1-78):

1. Create a JMS connection factory by instantiating a `JMSConnectionFactoryMBean`.

   The `CREATE -mbean` command creates an MBean and specifies the MBean’s `WebLogicObjectName`. (See "JMS Configuration Namespace" in *Programming WebLogic Management Services with JMX*.) In this object name:

   – `mymedrec:` specifies the name of the WebLogic Server domain.

   - `Name=MedRecQueueFactory` provides a unique name for the JMS connection factory you want to create.
   - `Type=JMSConnectionFactory` specifies the type of MBean to create.

2. Configure the JMS connection factory by setting attributes on the `JMSConnectionFactory` that you instantiated.

   In the `SET` command:

   - The `-mbean` argument specifies the `WebLogicObjectName` of the `JMSConnectionFactory` that you instantiated.
   - The `-property` argument specifies an MBean attribute name and its value.

     To see all attributes and legal values of the `JMSConnectionFactory`, refer to the Javadoc.

3. Create and configure a JMS store that uses JDBC and a database by instantiating and setting values on a `JMSJDBCStoreMBean`.

   The command that sets the value of the `ConnectionPool` attribute specifies the `WebLogicObjectName` of a JDBC connection pool because the `JMSJDBCStoreMBean` Javadoc indicates that the `ConnectionPool` attribute must be a `JDBCConnectionPoolMBean` object name:

   ```
   public void setConnectionPool(JDBCConnectionPoolMBean connectionPool)
   ```

4. Create and configure a JMS Server by instantiating and setting values on a `JMSServerMBean`.

   The command that sets the value of the `Store` attribute specifies the `WebLogicObjectName` of a JMS store because the `JMSServerMBean` Javadoc indicates that the `Store` attribute must be an object name of type `JMSStoreMBean`:

   ```
   public void setStore(JMSStoreMBean store)
   ```

   `JMSJDBCStore` is a subtype of `JMSStoreMBean`.

5. Create and configure three JMS queues by instantiating and setting values on three MBeans of type `JMSQueueMBean`.

   The queues are named `RegistrationQueue`, `MailQueue` and `XMLQueue`.

   To see all attributes and legal values of the `JMSQueueMBean`, refer to the Javadoc.

Creating Mail Resources

The commands in Listing 2-3 add email capabilities to the MedRec applications in the mymedrec domain by creating and configuring a `MailSessionMBean`. You can save the commands in a text file and invoke them with the `weblogic.Admin BATCHUPDATE` command (see "BATCHUPDATE" on page 1-78).

Note: To see all attributes and legal values of the `MailSessionMBean`, refer to the Javadoc. For more information about the WebLogic Server mail service, see "Mail" in the Administration Console Online Help.

Listing 2-3 Creating Mail Resources

```
CREATE -mbean mymedrec:Name=MedicalRecordsMailSession,Type=MailSession

SET -mbean mymedrec:Name=MedicalRecordsMailSession,Type=MailSession
-property JNDIName "mail/MedRecMailSession"

SET -mbean mymedrec:Name=MedicalRecordsMailSession,Type=MailSession
-property Properties "mail.user=joe;mail.host=mail.mycompany.com"
```

Creating and Configuring Managed Servers

In a production environment, BEA recommends that you create one or more Managed Servers to host applications and resources. Use the Administration Server only to configure and manage the domain.

The commands in Listing 2-4 create and configure a Managed Server in the mymedrec domain.

Listing 2-4 Creating and Configuring Managed Servers

```
# Creating and Configuring a Server
CREATE -mbean mymedrec:Name=MedRecServer1,Type=Server

SET -mbean mymedrec:Name=MedRecServer1,Type=Server
-property ListenPort "8011"

# Targeting Resources to MedRecServer1
SET -mbean mymedrec:Name=MedRecPoolXA,Type=JDBCConnectionPool
-property Targets "mymedrec:Name=myMedRecServer1,Type=Server"
```

```
SET -mbean mymedrec:Name=MedRecTxDataSource,Type=JDBCTxDataSource
-property Targets "mymedrec:Name=myMedRecServer1,Type=Server"

SET -mbean mymedrec:Name=myMedRecPoolXA,Type=JDBCConnectionPool
-property Targets "mymedrec:Name=myMedRecServer1,Type=Server"

SET -mbean mymedrec:Name=MedRecQueueFactory,Type=JMSConnectionFactory
-property Targets "mymedrec:Name=myMedRecServer1,Type=Server"

SET -mbean mymedrec:Name=MedRecJMSServer,Type=JMSServer
-property Targets "mymedrec:Name=myMedRecServer1,Type=Server"

SET -mbean mymedrec:Name=MedicalRecordsMailSession,Type=MailSession
-property Targets "mymedrec:Name=myMedRecServer1,Type=Server"
```

When invoked by the `weblogic.Admin BATCHUPDATE` command, the commands do the following (see "BATCHUPDATE" on page 1-78):

1. Create a server instance named myMedRecServer1 by instantiating a `ServerMBean`.

   The `CREATE -mbean` command creates an MBean and specifies the MBean's `WebLogicObjectName`.

2. Configure myMedRecServer1 to listen on port 8011 by setting the `ListenPort` attribute of myMedRecServer1's `ServerMBean`.

3. Make resources available to the myMedRecServer1 Managed Server by setting the `Targets` attribute of the resources' MBeans.

   For example, to make the myMedRecPoolXA JDBC connection pool available to myMedRecServer1, you set the `Targets` attribute of the myMedRecPoolXA's `JDBCConnectionPoolMBean`. The `JDBCConnectionPoolMBean` Javadoc indicates that the `Targets` attribute must be an object name of type `ServerMBean`:

   ```
   public void setTargets(TargetMBean[] Targets)
   ```

   `ServerMBean` is a subtype of `TargetMBean`. See "Using weblogic.Admin Commands to Target Resources" on page 2-29.

Configuring Machines and Node Manager Properties

The commands in Listing 2-5, configure Machines and Node Manager properties in the mymedrec domain.

This is a common task for domains that contain multiple server instances. By configuring machines and Node Manager properties for a server, the server can be started and managed by a Node Manager. See "Overview of Node Manager" in *Configuring and Managing WebLogic Server*.

Listing 2-5 Configuring Machines and Node Manager Properties

```
# Create a Machine
CREATE -mbean mymedrec:Name=WLSHost,Type=Machine

SET -mbean mymedrec:Machine=WLSHost,Name=WLSHost,Type=NodeManager
-property ListenAddress WLSHost -property ListenPort 5560

# Assign the Machine to a Server
SET  -mbean mymedrec:Name=myMedRecServer1,Type=Server
-property Machine "mymedrec:Name=WLSHost,Type=Machine"
```

When invoked by the `weblogic.Admin BATCHUPDATE` command, the commands do the following (see "BATCHUPDATE" on page 1-78):

1. Create a machine by instantiating a `MachineMBean`.

   The `CREATE -mbean` command creates an MBean and specifies the MBean's `WebLogicObjectName`. (See "Machines and Node Manager Configuration Namespace" in *Programming WebLogic Management Services with JMX*.) In this object name:

   – `mymedrec:` specifies the name of the WebLogic Server domain.

   – `Name=WLHost1` provides a unique name for the machine you want to create. By convention, the machine name reflects the name of the computer that the machine represents. See "Machines" in the Administration Console Online Help.

   – `Type=Machine` specifies the type of MBean to create.

2. Configure a `NodeManagerMBean` for the machine. When you create a `MachineMBean` instance, WebLogic Server also creates a `NodeManagerMBean` to specify the listen address, listen port, and security information that a server instance uses to communicate with a Node Manager running on a specific machine.

   The example commands set non-default values for the `ListenAddress` and `ListenPort` properties. In the `NodeManagerMBean` object name:

   – `mymedrec:` specifies the name of the WebLogic Server domain.

- `Machine=WLSHost` indicates that the `NodeManagerMBean` is a child of the WLSHost `MachineMBean`. (See "Machines and Node Manager Configuration Namespace" in *Programming WebLogic Management Services with JMX*.)
- `Name=WLHost1` follows the WebLogic Server convention of having child objects reflect the object name of the parent.
- `Type=NodeManager` specifies the type of MBean to create.

To see all attributes and legal values of the `NodeManagerMBean`, refer to the Javadoc.

3. Assign the machine to myMedRecServer1 by setting the `Machine` attribute of myMedRecServer1's `ServerMBean`. Assigning a machine to a server enables the server to be started by the machine's Node Manager.

   The `ServerMBean` Javadoc indicates that the `Machine` attribute must be an object name of type `MachineMBean`:

   ```
   public void setMachine(MachineMBean machine)
   ```

Deploying Applications

The `weblogic.Deployer` commands in Listing 2-6 deploy the sample MedRec applications in the mymedrec domain.

Listing 2-6 Deploying Applications

```
java weblogic.Deployer -url t3://localhost:8001 -username weblogic
-password weblogic -targets myMedRecServer1 -name MedRecEAR
-deploy D:\bea\weblogic81\samples\server\medrec\dist\medrecEar

java weblogic.Deployer -url t3://localhost:8001 -username weblogic
-password weblogic -targets myMedRecServer1 -name PhysicianEAR
-deploy D:\bea\weblogic81\samples\server\medrec\dist\physicianEar

java weblogic.Deployer -url t3://localhost:8001 -username weblogic
-password weblogic -targets myMedRecServer1 -name StartupEAR
-deploy D:\bea\weblogic81\samples\server\medrec\dist\startupEar
```

Notes:

- The `weblogic.Deployer` command does not provide an equivalent to the `weblogic.Admin BATCHUPDATE` command, so you must invoke `java weblogic.Deployer` separately for each application that you want to deploy.
- You must invoke these commands on the computer that hosts the Administration Server for the mymedrec domain.
- Because the MedRec applications use JDBC connection pools, you must specify the JDBC driver in the `CLASSPATH` environment variable. See "Setting Up the Environment" on page 2-3.
- The commands specify the mymedrec domain by providing the listen address for the Administration Server in the mymedrec domain (`localhost:8001`).
- In the sample commands, the application files are located in the `D:\bea\weblogic81\samples\server\medrec\src` directory.
- To deploy the applications onto the myMedRecServer1 Managed Server, include the `-targets myMedRecServer1` argument. If the myMedRecServer1 server instance is active when you invoke the `weblogic.Deployer` command, deployment initiates immediately. Otherwise, deployment initiates when you start the myMedRecServer1 server instance.

See "weblogic.Deployer Utility" in *Deploying WebLogic Server Applications*.

Sample Korn Shell Script

This section describes an example Korn shell script that creates and configures a domain as described in "Using Command-Line Utilities to Clone the MedRec Domain: Main Steps" on page 2-2.

To use the sample Korn shell script:

1. Save Listing 2-7 as a shell script named `CreateDomain.sh`.

   BEA recommends that you save this script in an empty directory. By default, the shell script creates a domain directory named `.\mymedrec`. That is, the `mymedrec` directory is a child of the directory from which you invoke the command.

2. Save Listing 2-8 in a text file named `ConfigDomain.txt` and locate it in the same directory as `CreateDomain.sh`.

3. Open a Korn shell and set up the environment as follows:

   a. Invoke *`WL_HOME`*`\common\bin\startPointBase.sh`.

This script starts a PointBase database that the sample domain uses.

b. Invoke *WL_HOME*\samples\domains\medrec\setMedRecEnv.sh.

This script adds the required WebLogic Server classes and PointBase classes to the CLASSPATH variable and other required programs to the PATH variable.

4. Invoke CreateDomain.sh.

The CreateDomain.sh script:

- Creates a domain and starts the domain's Administration Server.
- Determines that the mymedrec domain's Administration Server has successfully started by periodically invoking the weblogic.Admin CONNECT command.

  The CONNECT command connects to a WebLogic Server instance 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 1-40.
- Invokes weblogic.Admin BATCHUPDATE to configure resources.

5. To deploy the MedRec sample applications:

a. Change to the mymedrec directory that the CreateDomain.sh script created.

b. Invoke the following command:

```
java -Dweblogic.Name=myMedRecServer1
-Dweblogic.management.server=http://localhost:8001 weblogic.Server &
```

c. After the myMedRecServer1 instance starts, invoke the weblogic.Deployer commands described in "Deploying Applications" on page 2-12.

Listing 2-7 Korn Shell Script for Recreating the MedRec Domain

```
#!/bin/sh


#########################
# Defining Functions    #
#########################

# Wrapper function for weblogic.Admin
wladmin()
{
  java weblogic.Admin -username $ADMINUSER -password $ADMINPASS `
       -adminurl $ADMINHOST:$ASPORT $*
}

# This function waits for a server to boot, if server is running, sets
# SERVERSTATE var to "RUNNING", otherwise, "SHUTDOWN"
#
# Params:
#   $1    Wait interval in seconds
#   $2    Max wait period, in seconds
#

WaitForServer()
{
  WAIT_INTERVAL=${1:-10}
  MAX_WAIT=${2:-120}
  CONNECTCMD="wladmin connect 1"
  CONNECTRESPONSE=
  CONNECTED=

  let total_wait=0
  while test "$CONNECTED" != "Connection" && test total_wait -lt MAX_WAIT
  do
    sleep 10
    let "total_wait=$total_wait+$WAIT_INTERVAL"
    echo Attempting to connect to server...
    CONNECTRESPONSE="`$CONNECTCMD`"
    CONNECTED="`echo $CONNECTRESPONSE | cut -d: -f1`"
    echo "Response to connect command: $CONNECTRESPONSE"
  done

  # Set the status so the caller can check the result
  if [ "$CONNECTED" == "Connection" ]; then
    SERVERSTATE="RUNNING"
    echo Connected to server!
  else
    SERVERSTATE="SHUTDOWN"
    echo Could not connect to server
  fi
}

# This function creates a new domain and starts the Administration
# Server.
#
CreateDomain() {
 Echo "Starting server $SERVERNAME"
  java -Dweblogic.management.GenerateDefaultConfig=true `
    -Dweblogic.management.username=${ADMINUSER} `
    -Dweblogic.management.password=${ADMINPASS} `
    -Dweblogic.Domain=${DOMAIN} `
    -Dweblogic.Name=${SERVERNAME} `
    -Dweblogic.ListenPort=${ASPORT} `
    weblogic.Server > ${SERVERNAME}.out 2>&1 &

  WaitForServer 10 120
}

###########################
# Setting the Environment #
###########################
# Override values in the script with any
# values supplied from the command line.
if [ $# -gt 0 ]; then
  echo Setting environment variables $*
  export $*
fi

# Set script variables.
PROTOCOL=t3
ADMINUSER=${ADMINUSER:-weblogic}
ADMINPASS=${ADMINPASS:-weblogic}
ADMINHOST=${ADMINHOST:-localhost}
ASPORT=${ASPORT:-8001}

DOMAIN=${DOMAIN:-mymedrec}
DOMAINDIR=${DOMAINDIR:-$DOMAIN}
SERVERNAME=${SERVERNAME:-MedRecServer}

# Check for environment variables.
if [ -z "$WL_HOME" ]; then
  echo "Need to set WL_HOME first!"
  exit 1
fi

###########################
# Invoking the Functions  #
###########################

# Check to see if server is runningConnected
SERVERSTATE="" # reset result of Wait function
WaitForServer 5 5
if [ "$SERVERSTATE" == "RUNNING" ]; then
  echo "Server is already running on port $ASPORT!"
  exit 1
fi

# Create the domain directory.
mkdir ${DOMAINDIR}
status=$?
 if [ $status != 0 ]; then
```

```
    echo "Could not create domain directory ${DOMAINDIR}"
    exit 1
  fi

# CD to the domain directory and create the domain.
startDir=`pwd`
cd ${DOMAINDIR}
CreateDomain

#Return to starting location
cd $startDir

#Use weblogic.Admin BATCHUPDATE command to configure resources
wladmin BATCHUPDATE -batchFile ConfigDomain.txt -batchCMDVerbose `
-continueOnError
```

You can save the commands in Listing 2-8 in a text file named `ConfigDomain.txt` and invoke them with the `weblogic.Admin BATCHUPDATE` command in Listing 2-7. Make sure that each command is on a separate, single line. For example, "`CREATE_POOL myMedRecPoolXA driver="com.pointbase.xa.xaDataSource", url=jdbc:pointbase:server://localhost:9093/demo, props=user=medrec;password=medrec; DatabaseName=jdbc:pointbase:server://localhost:9093/demo,maxCapacity=10`" must be on a single line.

Listing 2-8 Input for BATCHUPDATE Command

```
# Commands for weblogic.Admin BATCHUPDATE

############################
# Creating JDBC Resources  #
############################

# Create a JDBC Connection Pool for Applications
CREATE_POOL myMedRecPoolXA
driver="com.pointbase.xa.xaDataSource",
url=jdbc:pointbase:server://localhost:9093/demo,
props=user=medrec;password=medrec;
DatabaseName=jdbc:pointbase:server://localhost:9093/demo,maxCapacity=10

# Create a Transactional Data Source
CREATE -mbean mymedrec:Name=MedRecTxDataSource,Type=JDBCTxDataSource
```

```
# Configure the Transactional Data Source
SET -mbean mymedrec:Name=MedRecTxDataSource,Type=JDBCTxDataSource
-property JNDIName "MedRecTxDataSource"

SET -mbean mymedrec:Name=MedRecTxDataSource,Type=JDBCTxDataSource
-property PoolName "myMedRecPoolXA"

# Create another JDBC Connection Pool for the JMS JDBC Store
CREATE_POOL myMedRecPool driver="com.pointbase.jdbc.jdbcUniversalDriver",
url=jdbc:pointbase:server://localhost:9093/demo,
props=user=medrec;password=medrec,
DatabaseName=jdbc:pointbase:server://localhost:9093/demomaxCapacity=10

############################
# Creating JMS Resources  #
############################

# Creating a JMS Connection Factory
CREATE -mbean mymedrec:Name=MedRecQueueFactory,Type=JMSConnectionFactory

# Configuring the JMS Connection Factory
SET -mbean mymedrec:Name=MedRecQueueFactory,Type=JMSConnectionFactory
-property JNDIName "jms/MedRecQueueConnectionFactory"

SET -mbean mymedrec:Name=MedRecQueueFactory,Type=JMSConnectionFactory
-property XAServerEnabled "true"

#
# Creating and Configuring a JMS JDBC Store
CREATE -mbean mymedrec:Name=MedRecJDBCStore,Type=JMSJDBCStore

SET -mbean mymedrec:Name=MedRecJDBCStore,Type=JMSJDBCStore
-property ConnectionPool "mymedrec:Name=myMedRecPool,Type=JDBCConnectionPool"

SET -mbean mymedrec:Name=MedRecJDBCStore,Type=JMSJDBCStore
-property PrefixName "MedRec"

# Creating and Configuring a JMS Server
CREATE -mbean mymedrec:Name=MedRecJMSServer,Type=JMSServer

SET -mbean mymedrec:Name=MedRecJMSServer,Type=JMSServer
-property Store "mymedrec:Name=MedRecJDBCStore,Type=JMSJDBCStore"

# Creating and Configuring a Queue
CREATE -mbean
mymedrec:JMSServer=MedRecJMSServer,Name=RegistrationQueue,Type=JMSQueue

SET -mbean
mymedrec:JMSServer=MedRecJMSServer,Name=RegistrationQueue,Type=JMSQueue
-property JNDIName "jms/REGISTRATION_MDB_QUEUE"
```

```
# Creating and Configuring an Additional Queue
CREATE -mbean mymedrec:JMSServer=MedRecJMSServer,Name=MailQueue,Type=JMSQueue

SET -mbean mymedrec:JMSServer=MedRecJMSServer,Name=MailQueue,Type=JMSQueue
-property JNDIName "jms/MAIL_MDB_QUEUE"

# Creating and Configuring an Additional Queue
CREATE -mbean mymedrec:JMSServer=MedRecJMSServer,Name=XMLQueue,Type=JMSQueue

SET -mbean mymedrec:JMSServer=MedRecJMSServer,Name=XMLQueue,Type=JMSQueue
-property JNDIName "jms/XML_UPLOAD_MDB_QUEUE"

############################
# Creating Mail Resources  #
############################
CREATE -mbean mymedrec:Name=MedicalRecordsMailSession,Type=MailSession

SET -mbean mymedrec:Name=MedicalRecordsMailSession,Type=MailSession
-property JNDIName "mail/MedRecMailSession"

SET -mbean mymedrec:Name=MedicalRecordsMailSession,Type=MailSession
-property Properties "mail.user=joe;mail.host=mail.mycompany.com"

#############################################
# Creating and Configuring a Managed Server #
#############################################

# Creating and Configuring a Server
CREATE -mbean mymedrec:Name=myMedRecServer1,Type=Server

SET -mbean mymedrec:Name=myMedRecServer1,Type=Server
-property ListenPort "8011"

# Targeting Resources to myMedRecServer1
SET -mbean mymedrec:Name=myMedRecPool,Type=JDBCConnectionPool
-property Targets "mymedrec:Name=myMedRecServer1,Type=Server"

SET -mbean mymedrec:Name=MedRecTxDataSource,Type=JDBCTxDataSource
-property Targets "mymedrec:Name=myMedRecServer1,Type=Server"

SET -mbean mymedrec:Name=myMedRecPoolXA,Type=JDBCConnectionPool
-property Targets "mymedrec:Name=myMedRecServer1,Type=Server"

SET -mbean mymedrec:Name=MedRecQueueFactory,Type=JMSConnectionFactory
-property Targets "mymedrec:Name=myMedRecServer1,Type=Server"

SET -mbean mymedrec:Name=MedRecJMSServer,Type=JMSServer
-property Targets "mymedrec:Name=myMedRecServer1,Type=Server"

SET -mbean mymedrec:Name=MedicalRecordsMailSession,Type=MailSession
-property Targets "mymedrec:Name=myMedRecServer1,Type=Server"
```

```
##############################################
# Configuring Machine and Node Manager Props #
##############################################

# Create a Machine
CREATE -mbean mymedrec:Name=WLSHost,Type=Machine

SET -mbean mymedrec:Machine=WLSHost,Name=WLSHost,Type=NodeManager
-property ListenAddress WLSHost -property ListenPort 5560

# Assign the Machine to a Server
SET  -mbean mymedrec:Name=myMedRecServer1,Type=Server
-property Machine "mymedrec:Name=WLSHost,Type=Machine"
```

Using weblogic.Admin Commands to Manage Users and Groups

In the WebLogic Security Service, an **Authentication provider** is the software component that proves the identity of users or system processes. An Authentication provider also remembers, transports, and makes that identity information available to various components of a system when needed. A security realm can use different types of Authentication providers to manage different sets of users and groups. See "Authentication Providers" in *Developing Security Providers for WebLogic Server.*

You can use the `weblogic.Admin` utility to invoke operations on the following types of Authentication providers:

- The WebLogic Authentication provider, `weblogic.management.security.authentication.AuthenticatorMBean`.

  By default, all security realms use this Authentication provider to manage users and groups.

- The LDAP Authentication providers that extend `weblogic.management.security.authentication.LDAPAuthenticatorMBean`.

- Custom Authentication providers that extend `weblogic.security.spi.AuthenticationProvider` and extend the optional Authentication SSPI MBeans. See "SSPI MBean Quick Reference" in *Developing Security Providers for WebLogic Server.*

The following sections describe basic tasks for managing users and groups with the `weblogic.Admin` utility:

- "Finding the Object Name for an AuthenticationProvider MBean" on page 2-21

- "Creating a User" on page 2-22
- "Adding a User to a Group" on page 2-22
- "Verifying Whether a User is a Member of a Group" on page 2-22
- "Listing Groups to Which a User Belongs" on page 2-23
- "Limiting Group Membership Searching in an LDAP Server" on page 2-23
- "Listing Users and Groups in a Security Realm" on page 2-24
- "Changing a Password" on page 2-27
- "Protecting User Accounts in a Security Realm" on page 2-27

For information about additional tasks that the `AuthenticationProvider` and the optional MBeans support, refer to the Javadoc for the `weblogic.management.security.authentication` package.

Finding the Object Name for an AuthenticationProvider MBean

To invoke operations on an Authentication provider, you must specify the object name of the provider's `AuthenticationProviderMBean`.

The object name of the WebLogic Authentication provider is:

`Security:Name=`*`realmName`*`DefaultAuthenticator`

where *`realmName`* is the name of a security realm. For example, `Security:Name=myrealmDefaultAuthenticator`.

The object name of the LDAP Authentication providers is:
`Security:Name=realmNameLDAPAuthenticator`

BEA recommends that you follow a similar convention when you create your own Authentication providers:

`Security:Name=`*`realmNameAuthenticatorName`*

If you use the Administration Console to add an Authentication provider to the realm, your `AuthenticationProviderMBean` name will follow the recommended naming convention.

To verify the object name of an `AuthenticationProviderMBean`, use the `weblogic.Admin QUERY` command. See "QUERY" on page 1-72.

Creating a User

To create a user, invoke the `UserEditorMBean.createUser` method, which is extended by the security realm's `AuthenticationProvider` MBean. See the Javadoc for the `createUser` method.

The method requires three input parameters:

`username password user-description`

Separate the parameters with a space. If any parameter contains a space, surround it with quotes.

The following example invokes `createUser` on the WebLogic Authentication provider:

```
java weblogic.Admin -adminurl localhost:8001 -username weblogic -password
weblogic invoke -mbean Security:Name=myrealmDefaultAuthenticator
-method createUser my-user1 mypassword "my user"
```

If the command succeeds, it prints `OK` to standard out.

Adding a User to a Group

To add a user to a group, invoke the `GroupEditorMBean.addMemberToGroup` method, which is extended by the security realm's `AuthenticationProvider` MBean. See the Javadoc for the `addMemberToGroup` method.

The method requires two input parameters:

`groupname username`

The following example invokes `addMemberToGroup` on the WebLogic Authentication provider:

```
java weblogic.Admin -adminurl localhost:8001 -username weblogic -password
weblogic invoke -mbean Security:Name=myrealmDefaultAuthenticator
-method addMemberToGroup Administrators my-user1
```

If the command succeeds, it prints `OK` to standard out.

Verifying Whether a User is a Member of a Group

To verify whether a user is a member of a group, invoke the `GroupEditorMBean.isMember` method, which is extended by the security realm's `AuthenticationProvider` MBean. See the Javadoc for the `isMember` method.

The method requires three input parameters:

`groupname username boolean`

where *boolean* specifies whether the command searches within child groups. If you specify `true`, the command returns `true` if the member belongs to the group that you specify or to any of the groups contained within that group.

The following example invokes `isMember` on the WebLogic Authentication provider:

```
java weblogic.Admin -adminurl localhost:8001 -username weblogic -password
weblogic invoke -mbean Security:Name=myrealmDefaultAuthenticator
-method isMember Administrators weblogic true
```

If the user is a member of the group, the command prints `true` to standard out.

Listing Groups to Which a User Belongs

To see a list of groups that contain a user or a group, invoke the `MemberGroupListerMBean.listMemberGroups` method, which is extended by the security realm's `AuthenticationProvider` MBean. See the Javadoc for the `listMemberGroups` method.

The method requires one input parameter:

memberUserOrGroupName

where *memberUserOrGroupName* specifies the name of an existing user or a group.

The following example invokes `listMemberGroups` on the WebLogic Authentication provider:

```
java weblogic.Admin -adminurl localhost:8001 -username weblogic -password
weblogic invoke -mbean Security:Name=myrealmDefaultAuthenticator
-method listMemberGroups my-user1
```

The method returns a cursor, which refers to a list of names. The `weblogic.management.utils.NameLister.haveCurrent`, `getCurrentName`, and `advance` methods iterate through the returned list and retrieve the name to which the current cursor position refers. See the Javadoc for the `weblogic.management.utils.NameLister` interface.

Limiting Group Membership Searching in an LDAP Server

Unlimited, recursive searches for group membership can take up considerable time and produce a performance bottleneck. To control the depth of group membership searches, you can set the value of `GroupMembershipSearching` on the WebLogic Authentication provider, LDAP Authentication providers, or any other Authentication provider that implements the `AuthenticationProvider` interface.

To limit group searches, set the value of GroupMembershipSearching on the WebLogic Authentication provider or LDAP Authentication providers to limited. The GroupMembershipSearching attribute has the following valid values:

```
unlimited limited
```

where limited specifies whether the command searches within one or more levels of a nested group hierarchy. If you specify a limited search, the MaxGroupMembershipSearchLevel attribute must be specified. The default is an unlimited search.

The MaxGroupMembershipSearchLevel attribute specifies how many levels of group membership to search. Valid values are 0 and positive integers, where 0 specifies that the command searches only within direct group memberships. A positive integer specifies the number of levels to search.

For example, when searching for membership in Group A, 0 indicates that only direct members of Group A will be found. If Group B is a member of Group A, the members of Group B will not be found by this search. If the attribute is set to 1, a search for membership in Group A will return direct members of Group A and any members of groups which are direct members of Group A. If Group B is a member of Group A, the members of Group B will also be found by this search. However, if Group C is a member of Group B, the members of Group C will not be found by this search.

See the Javadoc for GroupMembershipSearching and MaxGroupMembershipSearchLevel.

The following example gets the GroupMembershipSearching property on the WebLogic Authentication provider:

```
java weblogic.Admin -adminurl localhost:7001 -username weblogic -password
weblogic get -mbean Security:Name=myrealmDefaultAuthenticator -pretty
-property GroupMembershipSearching -commotype
```

Listing Users and Groups in a Security Realm

To see a list of user or group names, you invoke a series of methods, all of which are available through the AuthenticationProvider interface:

- The GroupReaderMBean.listGroups and UserReaderMBean.listUsers methods take two input parameters: a pattern of user or group names to search for, and the maximum number of names that you want to retrieve.

  Because a security realm can contain thousands (or more) of user and group names that match the pattern, the methods return a cursor, which refers to a list of names.

  See Javadoc for the listGroups method or the Javadoc for the listUsers method.

- The `weblogic.management.utils.NameLister.haveCurrent`, `getCurrentName`, and `advance` methods iterate through the returned list and retrieve the name to which the current cursor position refers. See the Javadoc for the `weblogic.management.utils.NameLister` interface.

  Because the `weblogic.Admin` utility does not provide the necessary control structures, you must wrap these methods in a script. See "Example: Korn Shell Script for Listing Users" on page 2-25.

- The `weblogic.management.utils.NameLister.close` method releases any server-side resources that are held on behalf of the list.

Example: Korn Shell Script for Listing Users

The Korn shell script in Listing 2-9 retrieves up to 10 user names in a security realm.

Listing 2-9 Unix Korn Shell Script for Listing All Users

```
#!/bin/ksh

HOST=localhost
PORT=8001
USER=weblogic
PASS=weblogic
PROTOCOL=   # can be blank, t3, t3s, http, https, etc.

if [ -z "$PROTOCOL" ]; then
   URL=$HOST:$PORT
else
   URL=$PROTOCOL://$HOST:$PORT
fi

Cursor=$(java weblogic.Admin -adminurl $URL -username $USER `
         -password $PASS invoke `
         -mbean Security:Name=myrealmDefaultAuthenticator `
         -method listUsers \* 10)

haveCurrent="true"

while [[ "$haveCurrent" = "true" ]]; do
    haveCurrent=$(java weblogic.Admin -adminurl $URL `
     -username $USER -password $PASS invoke `
     -mbean Security:Name=myrealmDefaultAuthenticator `
     -method haveCurrent $Cursor)

    if [[ "$haveCurrent" = "true" ]]; then
```

```
       Username=$(java weblogic.Admin -adminurl $URL `
            -username $USER -password $PASS invoke `
            -mbean Security:Name=myrealmDefaultAuthenticator `
            -method getCurrentName $Cursor)

        print $Username

        java weblogic.Admin -adminurl $URL -username $USER `
            -password $PASS invoke `
            -mbean Security:Name=myrealmDefaultAuthenticator `
            -method advance $Cursor

    else

       print "No more names in list."
       java weblogic.Admin -adminurl $URL -username $USER `
           -password $PASS invoke `
           -mbean Security:Name=myrealmDefaultAuthenticator `
           -method close $Cursor

      return

    fi

done
```

You can save the script in Listing 2-9 in a text file and invoke it from a Korn shell. When you invoke the script, it does the following:

1. Invokes the `UserReaderMBean.listUsers` method and assigns the returned cursor to a variable.

   Because the authentication provider extends these MBeans, the script invokes the methods on the security realm's `AuthenticationProvider` MBean.

2. Constructs a loop that does the following:

   a. Invokes the `weblogic.management.utils.NameLister.haveCurrent` method on the cursor variable to determine if the current cursor position refers to a name.

      If the cursor refers to a name, `haveCurrent` prints `true` to standard out.

   b. If `haveCurrent` prints `true`, invokes the `weblogic.management.utils.NameLister.getCurrentName` method.

      The `getCurrentName` method prints the name to which the cursor refers.

   c. Invokes the `weblogic.management.utils.NameLister.advance` method on the cursor, which causes the cursor to refer to the next name in the list.

3. Invokes the `weblogic.management.utils.NameLister.close` method on the cursor to free any server-side resources that are held on behalf of the list.

Changing a Password

To change a user's password, invoke the `UserPasswordEditorMBean.changeUserPassword` method, which is extended by the security realm's `AuthenticationProvider` MBean. See the Javadoc for the `changeUserPassword` method.

The method requires three input parameters:

`username old-password new-password`

The following example invokes `changeUserPassword` on the WebLogic Authentication provider:

```
java weblogic.Admin -adminurl localhost:8001 -username weblogic -password
weblogic invoke -mbean Security:Name=myrealmDefaultAuthenticator
-method changeUserPassword my-user1 mypassword my!password
```

If the command succeeds, it prints `OK` to standard out.

Protecting User Accounts in a Security Realm

Weblogic Server provides a set of attributes to protect user accounts from intruders. By default, these attributes are set for maximum protection. You can decrease the level of protection for user accounts. For example, you can increase the number of login attempts before a user account is locked, increase the time period in which invalid login attempts are made before locking the user account, or change the amount of time a user account is locked.

The `AuthenticationProvider` MBean does not extend methods that you use to protect user accounts. Instead, retrieve the `UserLockoutManagerMBean` and invoke its methods. See the Javadoc for the `UserLockoutManagerMBean` interface.

The following tasks provide examples for invoking `UserLockoutManagerMBean` methods:

- "Set Consecutive Invalid Login Attempts" on page 2-28
- "Unlock a User Account" on page 2-28

Set Consecutive Invalid Login Attempts

The following commands set the number of consecutive invalid login attempts before a user account is locked out:

1. To determine the object name for your domain's `UserLockoutManagerMBean`, enter the following command:

   ```
   java weblogic.Admin -adminurl localhost:8001 -username weblogic
   -password weblogic QUERY -pretty -pattern Security:*
   ```

   If the command succeeds, it prints output similar to the following abbreviated example:

   ```
   .
   .
   .
   ---------------------------
   MBeanName: "Security:Name=myrealmUserLockoutManager"
           InvalidLoginAttemptsTotalCount: 0
           InvalidLoginUsersHighCount: 0
           LockedUsersCurrentCount: 0
           LockoutCacheSize: 5
           LockoutDuration: 30
           LockoutEnabled: true
           LockoutGCThreshold: 400
           LockoutResetDuration: 5
           LockoutThreshold: 5
   .
   .
   .
   ```

2. To set the value of the `LockoutThreshold` attribute, enter the following command:

   ```
   java weblogic.Admin -adminurl localhost:8001 -username weblogic
   -password weblogic set -mbean Security:Name=myrealmUserLockoutManager
   -property LockoutThreshold 3
   ```

   If the command succeeds, it prints `OK` to standard out.

For more information about setting the value of MBean attributes, see "Commands for Managing WebLogic Server MBeans" on page 1-62.

Unlock a User Account

The following command unlocks a user account:

```
java weblogic.Admin -adminurl localhost:8001 -username weblogic
-password weblogic invoke -mbean Security:Name=myrealmUserLockoutManager
-method clearLockout my-user1
```

If the command succeeds, it prints `OK` to standard out.

Using weblogic.Admin Commands to Target Resources

In a WebLogic Server domain, many resources are defined at the domain level and then targeted to the server instances or clusters that will use the resource. For example, a JDBC connection pool is a domain-wide resource that can be targeted to many servers or clusters. Other resources, such as Web Servers and Network Channels, are defined at the level of a server instance. These resources are not targeted because they are already child resources of a server and they cannot be shared by other server instances.

Note: Instead of targeting J2EE modules (such as enterprise applications, Web applications, and EJBs) BEA recommends that you **deploy** them using the `weblogic.Deployer` utility. See "weblogic.Deployer Utility" in *Deploying WebLogic Server Applications*.

To target a resource to a server or cluster:

1. Determine the object name of the servers or clusters to which you want to target a resource:

   – To see the object names of all server instances in a domain, enter the following command:

   ```
   java weblogic.Admin -adminurl adminListenAddress:ListenPort
     -username username -password password
      GET -type Server
   ```

   where *`adminListenAddress:ListenPort`* is the listen address and listen port of the domain's Administration Server. The output includes the object name for all `ServerMBean` instances, as illustrated in the bold text in the following truncated output:

   ```
   MBeanName: "medrec:Name=MedRecServer,Type=Server"
           AcceptBacklog: 50
           AdministrationPort: 0
           .
           .
           .
   MBeanName: "medrec:Name=myMS1,Type=Server"
           AcceptBacklog: 50
           AdministrationPort: 0
           .
           .
           .
   ```

– To see the object names of all clusters in a domain, enter the following command:

```
java weblogic.Admin -adminurl adminListenAddress:ListenPort
  -username username -password password
   GET -type Cluster -property Name
```

where *adminListenAddress*:*ListenPort* is the listen address and listen port of the domain's Administration Server.

2. Determine the object name of the resource that you want to target by entering the following command:

```
java weblogic.Admin -adminurl adminListenAddress:ListenPort
  -username username -password password
   GET -type resource-MBean-type -property Targets
```

where *adminListenAddress*:*ListenPort* is the listen address and listen port of the domain's Administration Server and *resource-MBean-type* is the name of an MBean type described in Table 2-1.

For example, the following command returns a `JDBCConnectionPool` Administration MBean for all JDBC Connection Pools in the domain. The output includes the object name of the `MyPool` connection pool and indicates which servers or clusters have already been targeted to the resource:

```
java weblogic.Admin -adminurl localhost:8001
  -username weblogic -password weblogic
   GET -type JDBCConnectionPool -property Targets
```

In the medrec sample domain, the command returns the following, which indicates that the MedRecPool-Oracle is not targeted to a server or cluster:

```
MBeanName: "medrec:Name=MedRecPool-Oracle,Type=JDBCConnectionPool"
        Targets:
---------------------------
MBeanName: "medrec:Name=MedRecPool-PointBase,Type=JDBCConnectionPool"
        Targets: MedRecServer
```

3. The command that you use to target a resource to a server or cluster depends on the type of resource:

–
```
java weblogic.Admin -adminurl adminListenAddress:ListenPort
  -username username -password password
  INVOKE -mbean resource-object-name
  -method addTarget server-or-cluster-object-name
```

This command appends an object name to the `Targets` attribute of a resource. It is valid only for resources whose `Targets` attribute can contain multiple values.

–
```
java weblogic.Admin -adminurl adminListenAddress:ListenPort
  -username username -password password
  SET -mbean resource-object-name
  -property Targets server-or-cluster-object-name
```

  This command overwrites any existing values in the `Targets` attribute with the values that you specify. It is valid for any resource that includes a `Targets` attribute. For example:

```
INVOKE -mbean "examples:Name=MyPool,Type=JDBCConnectionPool"
-method addTarget "examples:Name=MS1,Type=Server"
```

  appends a server named MS2 to the list of targets for the MyPool JDBC connection pool. The following command replaces the current set of targets with a server named MS2:

```
SET -mbean "examples:Name=MyPool,Type=JDBCConnectionPool"
-property Targets "examples:Name=MS2,Type=Server"
```

–
```
java weblogic.Admin -adminurl adminListenAddress:ListenPort
  -username username -password password
  SET -mbean server-object-name
  -property resource resource-object-name
```

  This command is valid only for resources that are represented by an attribute within the `ServerMBean`.

Table 2-1 lists all resource types that can be targeted and provides a sample `weblogic.Admin` command to target the resource to the MS1 and a cluster named MedRecCluster.

Table 2-1 Resource MBeans

Resource Type	To Target a Server	To Target a Cluster

ClusterMBean	`SET -mbean`
	`"medrec:Name=MS1,`
	`Type=Server`
	`-property Cluster`
	`"medrec:Name=MedRecCluster,Type=Cluster"`
		not applicable

DomainLogFilterMBean	`SET -mbean`
	`"medrec:Name=MS1,`
	`Type=Server"`
	`-property DomainLogFilter`
	`"medrec:Name=MyDomainLogFilter,`
	`Type=DomainLogFilter"`
		not applicable

Table 2-1 Resource MBeans

Resource Type	To Target a Server	To Target a Cluster
ForeignJMSServerMBean	`INVOKE -mbean "medrec:Name=MyForeignJMSServer,Type=ForeignJMSServer" -method addTarget "medrec:Name=MS1,Type=Server"`	`INVOKE -mbean "medrec:Name=MyForeignJMSServer,Type=ForeignJMSServer" -method addTarget "medrec:Name=MedRecCluster,Type=Cluster"`
JDBCConnectionPoolMBean	`INVOKE -mbean "medrec:Name=MedRecPool, Type=JDBCConnectionPool" -method addTarget "medrec:Name=MS1, Type=Server"`	`INVOKE -mbean "medrec:Name=MedRecPool, Type=JDBCConnectionPool" -method addTarget "medrec:Name=MedRecCluster,Type=Cluster"`
JDBCDataSourceMBean	`INVOKE -mbean "medrec:Name=MyDataSource,Type=JDBCDataSource -method addTarget "medrec:Name=MS1,Type=Server"`	`INVOKE -mbean "medrec:Name=MyDataSource,Type=JDBCDataSource -method addTarget "medrec:Name=MedRecCluster,Type=Cluster"`
JDBCMultiPoolMBean	`INVOKE -mbean "medrec:Name=MyMultiPool, Type=JDBCMultiPool" -method addTarget "medrec:Name=MS1, Type=Server"`	`INVOKE -mbean "medrec:Name=MyMultiPool, Type=JDBCMultiPool" -method addTarget "medrec:Name=MedRecCluster,Type=Cluster"`
JDBCTxDataSourceMBean	`INVOKE -mbean "medrec:Name=MedRecTxDataSource,Type=JDBCTxDataSource -method addTarget "medrec:Name=MS1,Type=Server"`	`INVOKE -mbean "medrec:Name=MedRecTxDataSource,Type=JDBCTxDataSource -method addTarget "medrec:Name=MedRecCluster,Type=Cluster"`

Table 2-1 Resource MBeans

Resource Type	To Target a Server	To Target a Cluster
JMSConnectionFactoryMBean	`INVOKE -mbean "medrec:Name=jms/MedRecQueueConnectionFactory,Type=JMSConnectionFactory" -method addTarget "medrec:Name=MS1,Type=Server"`	`INVOKE -mbean "medrec:Name=jms/MedRecQueueConnectionFactory,Type=JMSConnectionFactory" -method addTarget "medrec:Name=MedRecCluster,Type=Cluster"`
JMSDistributedQueueMBean	not applicable Individual server instances can be specified in the MBean's `Members` attribute, but only clusters can be specified in the `Targets` attribute.	`INVOKE -mbean "medrec: Name=myDistributedQueue, Type=JMSDistributedQueue" -method addTarget "medrec:Name=MedRecCluster,Type=Cluster"`
JMSDistributedTopicMBean	not applicable Individual server instances can be specified in the MBean's `Members` attribute, but only clusters can be specified in the `Targets` attribute.	`INVOKE -mbean "medrec: Name=myDistributedTopic, Type=JMSDistributedTopic" -method addTarget "medrec:Name=MedRecCluster,Type=Cluster"`
JMSServerMBean	`SET -mbean "medrec:Name=MedRecJMSServer,Type=JMSServer" -property Targets "medrec:Name=MS1,Type=Server"`	not applicable
MailSessionMBean	`INVOKE -mbean "medrec:Name=mail/MedRecMailSession, Type=MailSession" -method addTarget "medrec:Name=MS1, Type=Server"`	`INVOKE -mbean "medrec:Name=mail/MedRecMailSession, Type=MailSession" -method addTarget "medrec:Name=MedRecCluster,Type=Cluster"`

Table 2-1 Resource MBeans

Resource Type	To Target a Server	To Target a Cluster
MachineMBean and its associated NodeManagerMBean	`SET -mbean "mymedrec:Name=MS1, Type=Server" -property Machine "mymedrec:Name=WLSHost, Type=Machine"`	not applicable
MessagingBridgeMBean	`INVOKE -mbean "medrec:Name=MyMessagingBridge,Type=MessagingBridge" -method addTarget "medrec:Name=MS1, Type=Server"`	`INVOKE -mbean "medrec:Name=MyMessagingBridge,Type=MessagingBridge" -method addTarget "medrec:Name=MedRecCluster,Type=Cluster"`
ShutdownClassMBean	`INVOKE -mbean "mymedrec:Name=MyShutdownClass,Type=ShutdownClass" -method addTarget "mymedrec:Name=MS1, Type=Server"`	`INVOKE -mbean "mymedrec:Name=MyShutdownClass,Type=ShutdownClass" -method addTarget "mymedrec:Name=MedRecCluster,Type=Cluster"`
StartupClassMBean	`INVOKE -mbean "mymedrec:Name=MyStartupClass,Type=StartupClass" -method addTarget "mymedrec:Name=MS1, Type=Server"`	`INVOKE -mbean "mymedrec:Name=MyStartupClass,Type=StartupClass" -method addTarget "mymedrec:Name=MedRecCluster,Type=Cluster"`
VirtualHostMBean	`INVOKE -mbean "medrec: Name=myVirtualHost, Type=VirtualHost" -method addTarget "medrec:Name=MS1, Type=Server"`	`INVOKE -mbean "medrec: Name=myVirtualHost, Type=VirtualHost" -method addTarget "medrec:Name=MedRecCluster,Type=Cluster"`

Table 2-1 Resource MBeans

Resource Type	To Target a Server	To Target a Cluster
WTCServerMBean	`INVOKE -mbean "medrec: Name=myWTC Server, Type=WTCServer" -method addTarget "medrec:Name=MS1, Type=Server"`	not applicable
XMLRegistry	`SET -mbean "medrec:Name=MS1, Type=Server" -property XMLRegistry "medrec:Name=MyXMLRegistry,Type=XMLRegistry"`	not applicable

Using weblogic.Admin Commands to Configure the WebLogic SNMP Agent

WebLogic Server can use Simple Network Management Protocol (SNMP) to communicate with enterprise-wide management systems. The WebLogic Server subsystem that gathers WebLogic management data, converts it to SNMP communication modules (trap notifications), and forwards the trap notifications to third-party SNMP management systems is called the WebLogic SNMP agent. For more information about using SNMP with WebLogic Server, refer to the WebLogic SNMP Management Guide.

The commands in Listing 2-10 enable the WebLogic SNMP agent and configure it to send notifications (traps) to SNMP manager software.

Listing 2-10 Commands to Configure the WebLogic SNMP Agent

```
## Enable SNMPAgent and select a port
set -type SNMPAgent -property Enabled true -property SNMPPort 161

## create an SNMPTrapDestination where you want to send the traps that the agent
 generates
create -mbean medrec:Name=myManager,SNMPAgent=medrec,Type=SNMPTrapDestination
```

```
## Select a port for the trap destination
set -mbean medrec:Name=myManager,SNMPAgent=medrec,Type=SNMPTrapDestination
 -property Port 165

## Create an SNMPAttribute change trap and set the required properties
create -mbean
medrec:Name=myAttrChange,SNMPAgent=medrec,Type=SNMPAttributeChange

set -type SNMPAttributeChange -property AttributeMBeanType ServerRuntime
-property AttributeName State -property EnabledServers
medrec:Name=MedRecServer,Type=Server

## create a SNMPGaugeMonitor and set the required properties
create -mbean medrec:Name=myGauge,SNMPAgent=medrec,Type=SNMPGaugeMonitor

set -type SNMPGaugeMonitor -property MonitoredMBeanType ExecuteQueueRuntime
-property MonitoredAttributeName ExecuteThreadCurrentIdleCount
-property EnabledServers medrec:Name=MedRecServer,Type=Server
-property PollingInterval 20 -property ThresholdHigh 3 -property ThresholdLow 1

## create a SNMPStringMonitor and set the required properties
create -mbean medrec:Name=myString,SNMPAgent=medrec,Type=SNMPStringMonitor

set -type SNMPStringMonitor -property MonitoredMBeanType Server
-property MonitoredAttributeName StartupMode -property EnabledServers
medrec:Name=MedRecServer,Type=Server -property NotifyMatch true
-property StringToCompare RUNNING -property PollingInterval 20

## create a SNMPCounterMonitor and set the required properties
create -mbean medrec:Name=myCounter,SNMPAgent=medrec,Type=SNMPCounterMonitor

set -type SNMPCounterMonitor -property MonitoredAttributeName
ServicedRequestTotalCount -property MonitoredMBeanType ExecuteQueueRuntime
-property EnabledServers medrec:Name=MedRecServer,Type=Server -property Modulus
10 -property Offset 3 -property Threshold 3 -property PollingInterval 20
```

When invoked by the `weblogic.Admin BATCHUPDATE` command, the commands do the following (see "BATCHUPDATE" on page 1-78):

- Enable the agent and configure it to listen for requests from other SNMP software on port 161. Because the commands do not specify a community name (password), the SNMP agent uses the default community name of `public`.
- Create and enable a trap destination, which provides information for connecting to SNMP manager software. When the SNMP agent sends a message to this trap destination, it uses port 165 and the default community name of `public`.

- Create and configure an Attribute Change trap, a String Monitor trap, a Gauge Monitor trap, and a Counter Monitor trap. See "WebLogic Trap Notifications" in the *WebLogic SNMP Management Guide*.

Using the Sample Commands

To configure the WebLogic SNMP Agent, do the following:

1. Start the MedRec domain and Administration Server by running the following script:
   `WL_HOME\samples\domains\medrec\startMedRecServer` (Windows)
   `WL_HOME/samples/domains/medrec/startMedRecServer.sh` (UNIX)

   Where `WL_HOME` is the directory in which you installed WebLogic Server.

2. In a command shell, enter the following command:
   `WL_HOME\server\bin\setWLSEnv.cmd` (Windows)
   `WL_HOME/server/bin/setWLSEnv.sh` (UNIX)

3. Copy the commands in Listing 2-10 and paste them into an empty text file. Make sure that each command is on a separate, single line. For example, "`create -mbean medrec:Name=myManager,SNMPAgent=medrec,Type=SNMPTrapDestination`" must be on a single line.

4. Save the text file.

5. In the command shell, enter the following command:

   ```
   java weblogic.Admin -url localhost:8001 -username weblogic -password
   weblogic BATCHUPDATE -batchFile filename -continueonerror
   -batchCmdVerbose
   ```

   where *`filename`* is the name of the file that you created in step 4.

   Note: The above command assumes that you are running MedRecServer and the `BATCHUPDATE` command on the same Windows computer, and that the server is listening on port 8001. If you specified some other listen address or listen port, use the `-url` argument to specify your address and listen port.

   The `BATCHUPDATE` command returns `OK` for each command that it successfully runs.

To verify that you successfully configured the WebLogic SNMP agent:

1. Restart the Administration Server.

2. In a command shell, enter the following command:
   `WL_HOME\server\bin\setWLSEnv.cmd` (Windows)
   `WL_HOME/server/bin/setWLSEnv.sh` (UNIX)

3. In the same command shell, enter the following command:
   `java snmptrapd -p 165`

   This command starts a daemon that listens for SNMP traps on port 165. As the WebLogic SNMP agent on the MedRecServer sends traps to the trap destination that you defined, the trap daemon prints each trap to standard out.

   For more information about the `snmptrapd` command, refer to "snmptrapd" on page 6-13.

Using weblogic.Admin Commands to Create a Simple Cluster

The `weblogic.Admin BATCHUPDATE` command runs a sequence of `weblogic.Admin` commands that you specify in a text file. This section describes how to use `BATCHUPDATE` to create a simple example cluster. In this example cluster, all server instances run on the same WebLogic Server host as the Administration Server.

Before you can instantiate a cluster, your WebLogic Server license must include a cluster license. If you do not have a cluster license, contact your BEA sales representative. For more information about creating clusters, refer to "Setting Up Clusters" in the *Using WebLogic Server Clusters* guide.

To use `BATCHUPDATE` to create a simple cluster in the MedRec domain, do the following:

1. Start the MedRec domain and Administration Server. For example, you can open a command shell and run the following script:
   `*WL_HOME*\samples\domains\medrec\startMedRecServer` (Windows)
   `*WL_HOME*/samples/domains/medrec/startMedRecServer.sh` (UNIX)

   Where *`WL_HOME`* is the directory in which you installed WebLogic Server.

2. In a command shell, enter the following command:
   `*WL_HOME*\server\bin\setWLSEnv.cmd` (Windows)
   `*WL_HOME*/server/bin/setWLSEnv.sh` (UNIX)

3. Copy the commands in Listing 2-11 and paste them into an empty text file. Make sure that each command is on a separate, single line. For example, "`SET -mbean MedRec:Type=WebServer,Name=MedRecMS1,Server=MedRecMS1 -property LoggingEnabled true`" must be on a single line.

4. Save the text file.

5. Edit the commands that you pasted into the text file as follows:

- In the command `CREATE -mbean MedRec:Type=Machine,Name=calamine`, change **`calamine`** to refer either to the name or IP address of the computer that is running the Administration Server.
- If the listen ports 7777 and 7778 are already in use, change the port numbers in the commands.
- If the IP address `239.0.0.32` is already in use, change the address to a valid multicast address. For information about multicast addresses, refer to "Communications in a Cluster" in the *Using WebLogic Server Clusters* guide.

6. In the command shell, enter the following command:

```
java weblogic.Admin -url localhost:8001 -username weblogic -password
weblogic BATCHUPDATE -batchFile filename -continueonerror
-batchCmdVerbose
```

where *`filename`* is the name of the file that you created in step 4.

Note: The above command assumes that you are running the MedRec server and the `BATCHUPDATE` command on the same Windows computer, and that the server is listening on port 8001. If you specified some other listen address or listen port, use the `-url` argument to specify your address and listen port.

The `BATCHUPDATE` command returns `OK` for each command that it successfully runs.

To verify that you successfully created a cluster, view the Administration Console. In the left pane of the Administration Console, open the Cluster folder and make sure that it contains a cluster named MedRecCluster. Make sure that the cluster contains two server instances named MedRecMS1 and MedRecMS2. Also verify that the servers are targeted for the machine that the command `CREATE -mbean MedRec:Type=Machine,Name=calamine` (from Listing 2-11) creates.

Listing 2-11 BATCHUPDATE Commands for Creating a Cluster

```
#Create Server Instances
CREATE -mbean medrec:Type=Server,Name=MedRecMS1
CREATE -mbean medrec:Type=Server,Name=MedRecMS2

#Configure Servers
SET -mbean medrec:Type=Server,Name=MedRecMS1 -property ListenPort 7777
SET -mbean medrec:Type=WebServer,Name=MedRecMS1,Server=MedRecMS1 -property
LoggingEnabled true
SET -mbean medrec:Type=Server,Name=MedRecMS2 -property ListenPort 7778
SET -mbean medrec:Type=WebServer,Name=MedRecMS2,Server=MedRecMS2 -property
LoggingEnabled true
```

```
#Create and Configure Cluster
CREATE -mbean medrec:Type=Cluster,Name=MedRecCluster
SET -mbean medrec:Type=Cluster,Name=MedRecCluster -property MulticastAddress
239.0.0.32
SET -mbean medrec:Type=Server,Name=MedRecMS1 -property Cluster
medrec:Name=MedRecCluster,Type=Cluster
SET -mbean medrec:Type=Server,Name=MedRecMS2 -property Cluster
medrec:Name=MedRecCluster,Type=Cluster

#Target Resources to Cluster
#The MedRec domain provides a set of JDBC Connection Pools and Data Sources
#that must be deployed to all servers in the new cluster.
SET -mbean medrec:Name=MedRecPool-PointBase,Type=JDBCConnectionPool -property
Targets medrec:Name=MedRecCluster,Type=Cluster
SET -mbean medrec:Name=MedRecTxDataSource,Type=JDBCTxDataSource -property
Targets medrec:Name=MedRecCluster,Type=Cluster

SET -mbean
medrec:Name=jms/MedRecQueueConnectionFactory,Type=JMSConnectionFactory
 -property Targets medrec:Name=MedRecCluster,Type=Cluster

#Configure Machines and Node Manager
CREATE -mbean medrec:Type=Machine,Name=calamine
CREATE -mbean medrec:Type=NodeManager,Name=calamine
SET -mbean medrec:Type=Server,Name=MedRecMS1 -property Machine
medrec:Name=calamine,Type=Machine
SET -mbean medrec:Type=Server,Name=MedRecMS2 -property Machine
medrec:Name=calamine,Type=Machine

#Creating the Server MBeans also creates ServerStart MBeans, which configure
#the server to be started by a Node Manager. The next commands set values for
#the ServerStart MBeans.
SET -mbean medrec:Name=MedRecMS1,Server=MedRecMS1,Type=ServerStart -property
Username weblogic
SET -mbean medrec:Name=MedRecMS1,Server=MedRecMS1,Type=ServerStart -property
Password weblogic

SET -mbean medrec:Name=MedRecMS2,Server=MedRecMS2,Type=ServerStart -property
Username weblogic
SET -mbean medrec:Type=ServerStart,Name=MedRecMS2,Server=MedRecMS2 -property
Password weblogic
```

Deploying Applications and Starting Servers in the Simple Cluster

After you create the cluster, you can use either the Administration Server or the `weblogic.Deployer` utility to deploy applications to the cluster. For more information, refer to "Deployment Tools Reference" in the *Deploying WebLogic Server Applications* guide and "Deploying Applications and Modules" in the Administration Console Online Help.

The commands in Listing 2-11 enable the MedRecMS1 and MedRecMS2 server instances to be started by the Node Manager. For information on setting up Managed Servers to be started by a Node Manager, refer to the following sections in the *Configuring and Managing WebLogic Server* guide:

- "Configure a Machine to Use Node Manager"
- "Configure Managed Server Startup Arguments"
- "Starting and Stopping Node Manager"

Using the WebLogic Server Java Utilities

WebLogic Server provides a number of Java utilities and Ant tasks for performing administrative and programming tasks.

To use these utilities and tasks, you must set your `CLASSPATH` correctly. For more information, see "Modifying the Classpath" on page 4-2.

WebLogic Server provides several Java programs that simplify installation and configuration tasks, provide services, and offer convenient shortcuts. The Java utilities provided with WebLogic Server are all described below. The command-line syntax is specified for all utilities and, for some, examples are provided.

WebLogic Server also provides a number of Ant tasks that automate common application server programming tasks. The Apache Web site provides other useful Ant tasks as well, including tasks for packaging EAR, WAR, and JAR files. For more information, see http://jakarta.apache.org/ant/manual/.

- "appc" on page 3-3
- "AppletArchiver" on page 3-3
- "autotype" on page 3-4
- "BuildXMLGen" on page 3-4
- "CertGen" on page 3-4
- "ClientDeployer" on page 3-6

- “clientgen” on page 3-7
- “Conversion” on page 3-7
- “der2pem” on page 3-7
- “dbping” on page 3-8
- “DDInit” on page 3-12
- “Deployer” on page 3-15
- “ejbc” on page 3-15
- “EJBGen” on page 3-15
- “encrypt” on page 3-15
- “getProperty” on page 3-17
- “host2ior” on page 3-18
- “ImportPrivateKey” on page 3-18
- “jspc” on page 3-21
- “logToZip” on page 3-21
- “MBean Commands” on page 3-22
- “MulticastTest” on page 3-22
- “myip” on page 3-24
- “pem2der” on page 3-25
- “Pointbase and the Pointbase ant Task” on page 3-25
- “rmic” on page 3-26
- “Schema” on page 3-26
- “showLicenses” on page 3-28
- “source2wsdd” on page 3-28
- “system” on page 3-28
- “ValidateCertChain” on page 3-29

- “verboseToZip” on page 3-30
- “wlappc” on page 3-30
- “wlcompile” on page 3-31
- “wlconfig” on page 3-31
- “wldeploy” on page 3-31
- “wlpackage” on page 3-31
- “wlserver” on page 3-31
- “writeLicense” on page 3-31
- “wsdl2Service” on page 3-33
- “wsdlgen” on page 3-34
- “wspackage” on page 3-34

appc

The appc compiler generates and compiles the classes needed to deploy EJBs and JSPs to WebLogic Server. It also validates the deployment descriptors for compliance with the current specifications at both the individual module level and the application level. See appc in *Programming Weblogic Server Enterprise JavaBeans*.

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.

autotype

Use the `autotype` Ant task to generate non-built-in data type components, such as the serialization class, for Web services. The fully qualified name for the `autotype` Ant task is `weblogic.ant.taskdefs.webservices.javaschema.JavaSchema`. See Running the autotype Ant Task in *Programming WebLogic Web Services*.

BuildXMLGen

Use BuildXMLGen to generate a `build.xml` file for enterprise applications in the split-directory structure. For complete documentation of this utility, see Developing Applications Using the Split Development Directory Structure in *Developing WebLogic Server Applications*.

CertGen

The `CertGen` utility generates certificates that should only be used for demonstration or testing purposes and not in a production environment.

Syntax

```
$ java utils.CertGen

[-cacert <ca_cert_filename>] [-cakey <ca_key_filename>]
[-cakeypass <ca_key_password>] [-selfsigned]
[-certfile <certfile>] [-keyfile <privatekeyfile>]
[-keyfilepass <keyfilepassword>] [-strength <keystrength>]
[-cn <commonname>] [-ou <orgunit>] [-o <organization>]
[-l <locality>] [-s <state>] [-c <countrycode>]
[-subjectkeyid <subjectkeyidentifier>]
[-subjectkeyidformat UTF-8|BASE64]
```

Argument	Definition
ca_cert_filename	The file name of the issuer's CA public certificate.
ca_key_filename	The file name of the issuer's CA private key.
ca_key_password	The password for the issuer's CA private key.
selfsigned	Generates a self-signed certificate that can be used as a trusted CA certificate. If this argument is specified, the *ca_cert_filename*, *ca_key_filename*, and *ca_key_password* arguments should not be specified.
certfile	The name of the generated certificate file.
privatekeyfile	The name of the generated private key file.
keyfilepassword	The password for the private key.
keystrength	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.
commonname	The name to be associated with the generated certificate.
orgunit	The name of the organizational unit associated with the generated certificate.
organization	The name of the organization associated with the generated certificate.
locality	The name of a city or town.
state	The name of the state or province in which the organizational unit (ou) operates if your organization is in the United States or Canada, respectively. Do not abbreviate.
countrycode	Two-letter ISO code for your country. The code for the United States is US.
subjectkeyidentifier	Generates a certificate with the Subject Key identifier extension and the ID value specified on the command line.
UTF-8|BASE64	Format of the subjectkeyid value. Allowed values are UTF-8 or BASE64, with UTF-8 assumed by default.

Example

By default, the CertGen utility looks for the `CertGenCA.der` and `CertGenCAKey.der` files in the current directory, or in the *`WL_HOME`* directory, as specified in the `weblogic.home` system property or the `CLASSPATH`. Alternatively, you can specify CA files on the command line.

Enter the following command to generate certificate files named `testcert` with private key files named `testkey`:

```
$ java utils.CertGen mykeypass testcert testkey
Creating Domestic Key Strength - 1024

Encoding
................................................................
................................................................
................................................................
Created Private Key files - testkey.der and testkey.pem
Encoding
................................................................
................................................................
................................................................
Created Certificate files - testcert.der and testcert.pem
................................................................
```

ClientDeployer

You use `weblogic.ClientDeployer` to extract the client-side JAR file from a J2EE EAR file, creating a deployable JAR file. The `weblogic.ClientDeployer` class is executed on the Java command line with the following syntax:

```
java weblogic.ClientDeployer ear-file client
```

The *`ear-file`* argument is an expanded directory (or Java archive file with a `.ear` extension) that contains one or more client application JAR files.

For example:

```
java weblogic.ClientDeployer app.ear myclient
```

where `app.ear` is the EAR file that contains a J2EE client packaged in `myclient.jar`.

Once the client-side JAR file is extracted from the EAR file, use the `weblogic.j2eeclient.Main` utility to bootstrap the client-side application and point it to a WebLogic Server instance as follows:

java weblogic.j2eeclient.Main clientjar URL [application args]

For example

```
java weblogic.j2eeclient.Main helloWorld.jar t3://localhost:7001
Greetings
```

clientgen

Use `clientgen` to generate a client JAR file. See Running the clientgen Ant Task in *Programming WebLogic Web Services.*

Conversion

If you have used a pre-6.0 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 and has the same filename as the source `.der` file.

Syntax

```
$ java utils.der2pem derFile [headerFile] [footerFile]
```

Argument	Description
derFile	The name of the file to convert. The filename must end with a `.der` extension, and must contain a valid certificate in `.der` format.
headerFile	The header to place in the PEM file. The default header is “-----BEGIN CERTIFICATE-----”. Use a header file if the DER file being converted is a private key file, and create the header file containing one of the following: • “-----BEGIN RSA PRIVATE KEY-----” for an unencrypted private key. • “-----BEGIN ENCRYPTED PRIVATE KEY-----” for an encrypted private key. **Note:** There must be a new line at the end of the header line in the file.
footerFile	The header to place in the PEM file. The default header is “-----END CERTIFICATE-----”. Use a footer file if the DER file being converted is a private key file, and create the footer file containing one of the following in the header: • “-----END RSA PRIVATE KEY-----” for an unencrypted private key. • “-----END ENCRYPTED PRIVATE KEY-----” for an encrypted private key. **Note:** There must be a new line at the end of the header line in the file.

Example

```
$ java utils.der2pem graceland_org.der
Decoding
................................................................
```

dbping

The `dbping` command-line utility tests the connection between a DBMS and your client machine via a JDBC driver. You must complete the installation of the driver before attempting to use this utility. For more information on how to install a driver, see the documentation from your driver vendor. Also see Setting the Environment for a Type-4 Third-Party JDBC Driver in *Programming WebLogic JDBC*.

Creating a DB2 Package with dbping

With the WebLogic Type 4 JDBC Driver for DB2, you can also use the `dbping` utility to create a package on the DB2 server. When you ping the database with the dbping utility, the driver automatically creates the default package on the database server if it does not already exist. If the default package already exists on the database server, the `dbping` utility uses the existing package.

The default DB2 package includes 200 dynamic sections. You can specify a different number of dynamic sections to create in the DB2 package with the `-d` option. The `-d` option also sets `CreateDefaultPackage=true` and `ReplacePackage=true` on the connection used in the connection test, which forces the DB2 driver to replace the DB2 package on the DB2 server. (See DB2 Connection Properties for more information.) You can use the `-d` option with dynamic sections set at `200` to forcibly recreate a default package on the DB2 server.

Notes: When you specify the `-d` option, the `dbping` utility *recreates* the default package and uses the value you specify for the number of dynamic sections. It does not modify the existing package.

To create a DB2 package, the user that you specify must have CREATE PACKAGE privileges on the database.

Syntax

```
$ java utils.dbping DBMS [-d dynamicSections] user password DB
```

Argument	Definition
DBMS	Varies by DBMS and JDBC driver: DB2B—WebLogic Type 4 JDBC Driver for DB2 JCONN2—Sybase JConnect 5.5 (JDBC 2.0) driver JCONNECT—Sybase JConnect driver INFORMIXB—WebLogic Type 4 JDBC Driver for Informix MSSQLSERVER4—WebLogic jDriver for Microsoft SQL Server MSSQLSERVERB—WebLogic Type 4 JDBC Driver for Microsoft SQL Server ORACLE—WebLogic jDriver for Oracle ORACLEB—WebLogic Type 4 JDBC Driver for Oracle ORACLE_THIN—Oracle Thin Driver POINTBASE—PointBase Universal Driver SYBASEB—WebLogic Type 4 JDBC Driver for Sybase
[-d *dynamicSections*]	Specifies the number of dynamic sections to create in the DB2 package. This option is for use with the WebLogic Type 4 JDBC Driver for DB2 only. If the -d option is specified, the driver automatically sets CreateDefaultPackage=true and ReplacePackage=true on the connection and creates a DB2 package with the number of dynamic sections specified.
user	Valid database username for login. Use the same values you use with isql, sqlplus, or other SQL command-line tools. For DB2 with the -d option, the user must have CREATE PACKAGE privileges on the database.

Argument	Definition
password	Valid database password for the user. Use the same values you use with `isql` or `sqlplus`.
DB	Name and location of the database. Use the following format, depending on which JDBC driver you use: DB2B—Host:Port/DBName JCONN2—Host:Port/DBName JCONNECT—Host:Port/DBName INFORMIXB—Host:Port/DBName/InformixServer MSSQLSERVER4—Host:Port/DBName or [DBName@]Host[:Port] MSSQLSERVERB—Host:Port/DBName ORACLE—DBName (as listed in tnsnames.ora) ORACLEB—Host:Port/DBName ORACLE_THIN—Host:Port/DBName POINTBASE—Host[:Port]/DBName SYBASEB—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. • *InformixServer* is an Informix-specific environment variable that identifies the Informix DBMS server.

Example

```
C:\>java utils.dbping ORACLE_THIN scott tiger dbserver1:1561:demo


**** Success!!! ****


You can connect to the database in your app using:


java.util.Properties props = new java.util.Properties();
```

```
props.put("user", "scott");
props.put("password", "tiger");
props.put("dll", "ocijdbc9");
props.put("protocol", "thin");
java.sql.Driver d =
  Class.forName("oracle.jdbc.driver.OracleDriver").newInstance();
java.sql.Connection conn =
  Driver.connect("jdbc:oracle:thin:@dbserver1:1561:demo", props);
```

ddcreate

This Ant task calls EARInit, which generates an application.xml and a weblogic-application.xml file for an EAR. For more information, see "EarInit" on page 3-13.

DDInit

DDInit is a utility for generating deployment descriptors for applications to be deployed on WebLogic Server. Target a module's archive or folder and DDInit uses information from the module's class files to create appropriate deployment descriptor files.

WebLogic Builder, the graphical user interface for generating and editing deployment descriptors, runs DDInit to generate deployment descriptors. See WebLogic Builder for more information.

In its command-line version, unlike in WebLogic Builder, DDInit writes new files that overwrite existing descriptor files. If META-INF (for EAR or EJB), or WEB-INF (for Web Applications), does not exist, DDInit creates it.

Specify the type of J2EE deployable for which you want deployment descriptors generated by using the DDInit command specific to the type, as described below.

EJBInit

Target a JAR file or a folder containing files that you intend to archive as a JAR file, and EJBInit will generate the ejb-jar.xml and the weblogic-ejb-jar.xml files for the module.

```
java weblogic.marathon.ddinit.EJBInit <module>
```

EJBInit looks in folders under the target and finds EJBs (bean class, local or remote home, remote or local interface). Matches interfaces with beans, and determines from that match which home belongs to which bean. In the bean itself it looks for CMP fields. then for relationships between entity beans. From information gathered in this way, EJBInit writes the deployment descriptors.

`DDInit` supports EJB 2.0. DDInit will provide accurate results for session beans from 1.1, but is not likely to work for EJB 1.1 entity beans.

WebInit

Target a `WAR` file or a folder containing files that you intend to archive as a `WAR` file, and `WebInit` will create `web.xml` and `weblogic.xml` files for the module.

```
java weblogic.marathon.ddinit.WebInit <module>
```

EarInit

Generate an `application.xml` and a `weblogic-application.xml` file for an `EAR` using this command. Target an existing `EAR` or a folder containing `JAR` or `WAR` files you intend to archive into an `EAR` file.

```
java weblogic.marathon.ddinit.EarInit <module>
```

In WebLogic Builder, `EarInit looks` recursively at the entire tree under the targeted module. On the command line, you need to already have descriptors for the modules contained in the `EAR`. `application.xml` will account for the modules. The generated `weblogic-application.xml` will be an empty placeholder.

Limitations

- Generated XML representations of relations among entity beans are only accurate for one-to-one relations. For entity beans that already have descriptors, Builder does not refresh relations that have a “many” side.
- Support for generating descriptors for EJB 1.1 beans is not guaranteed; focus is on EJB 2.0

Example

This output from this example describes building deployment descriptor files for `ejb_st.jar`.

```
D:\dev\smarticket5\smarticket\bin>java weblogic.marathon.ddinit.EJBInit
ejb_st.jar
```

```
Found 4 classes that implement the EnterpriseBean interface
```

```
Discovered module type for D:\dev\smarticket5\smarticket\bin\ejb_st.jar

Found EJB components. Initializing descriptors

Creating desc for bean
com.sun.j2ee.blueprints.smarticket.ejb.customer.CustomerEJB

   *** found remote home:
com.sun.j2ee.blueprints.smarticket.ejb.customer.CustomerHome

   *** found remote interface:
com.sun.j2ee.blueprints.smarticket.ejb.customer.Customer

  Setting prim-key-class to 'java.lang.String'

Adding Entity bean 'CustomerEJB'

Creating desc for bean
com.sun.j2ee.blueprints.smarticket.ejb.localeinfo.LocaleInfoEJB

   *** found remote home:
com.sun.j2ee.blueprints.smarticket.ejb.localeinfo.LocaleInfoHome

   *** found remote interface:
com.sun.j2ee.blueprints.smarticket.ejb.localeinfo.LocaleInfo

LocaleInfoEJB is a Stateless Session bean

Adding Session bean 'LocaleInfoEJB'

Creating desc for bean
com.sun.j2ee.blueprints.smarticket.ejb.movieinfo.MovieInfoEJB

   *** found remote home:
com.sun.j2ee.blueprints.smarticket.ejb.movieinfo.MovieInfoHome

   *** found remote interface:
com.sun.j2ee.blueprints.smarticket.ejb.movieinfo.MovieInfo

MovieInfoEJB is a Stateless Session bean

Adding Session bean 'MovieInfoEJB'

Creating desc for bean
com.sun.j2ee.blueprints.smarticket.ejb.ticketsales.TicketSalesEJB

   *** found remote home:
com.sun.j2ee.blueprints.smarticket.ejb.ticketsales.TicketSalesHome

   *** found remote interface:
com.sun.j2ee.blueprints.smarticket.ejb.ticketsales.TicketSales
```

```
TicketSalesEJB is a Stateful Session bean

Adding Session bean 'TicketSalesEJB'

Writing descriptors

Building module with newly created descriptors

Finished building module
```

Deployer

Using the `weblogic.Deployer` tool, you can deploy J2EE applications and components to WebLogic Servers in a command-line or scripting environment. For detailed information on using this tool, see weblogic.Deployer Utility.

The `weblogic.Deployer` utility replaces the `weblogic.deploy` utility, which has been deprecated.

ejbc

Deprecated. See ejbc in *Programming Weblogic Server Enterprise JavaBeans*.

EJBGen

EJBGen is an Enterprise JavaBeans 2.0 code generator. You can annotate your Bean class file with javadoc tags and then use EJBGen to generate the Remote and Home classes and the deployment descriptor files for an EJB application, reducing to one the number of EJB files you need to edit and maintain.

See EJBGen Reference in *Programming WebLogic Server Enterprise JavaBeans*.

encrypt

The `weblogic.security.Encrypt` utility encrypts cleartext strings for use with WebLogic Server. The utility uses the encryption service of the current directory, or the encryption service for a specified WebLogic Server domain root directory.

Note: An encrypted string must have been encrypted by the encryption service in the WebLogic Server domain where it will be used. If not, the server will not be able to decrypt the string.

You can only run the `weblogic.security.Encrypt` utility on a machine that has at least one server instance in a WebLogic Server domain; it cannot be run from a client.

Note: BEA Systems recommends running the utility in the Administration Server domain directory or on the machine hosting the Administration Server and specifying a domain root directory.

Syntax

```
java   [ -Dweblogic.RootDirectory=dirname ]
       [ -Dweblogic.management.allowPasswordEcho=true ]
       weblogic.security.Encrypt [ password ]
```

Argument	Definition
`weblogic.RootDirectory`	Optional. WebLogic Server domain directory in which the encrypted string will be used. If not specified, the default domain root directory is the current directory (the directory in which the utility is being run).
`weblogic.management.allowPasswordEcho`	Optional. Allows echoing characters entered on the command line. `weblogic.security.Encrypt` expects that no-echo is available; if no-echo is not available, set this property to `true`.
`password`	Optional. Cleartext string to be encrypted. If omitted from the command line, you will be prompted to enter a password.

Examples

The utility returns an encrypted string using the encryption service of the domain located in the current directory.

```
java weblogic.security.Encrypt xxxxxx
```

```
{3DES}Rd39isn4LLuF884Ns
```

The utility returns an encrypted string using the encryption service of the specified domain location.

```
java -Dweblogic.RootDirectory=./mydomain weblogic.security.Encrypt xxxxxx
```

```
{3DES}hsikci118SKFnnw
```

The utility returns an encrypted string in the current directory, without echoing the password.

```
java weblogic.security.Encrypt
```

```
Password:
{3DES}12hsIIn56KKKs3
```

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:\
```

host2ior

The host2ior utility obtains the Interoperable Object Reference (IOR) of a WebLogic Server.

Syntax

```
$ java utils.host2ior hostname port
```

ImportPrivateKey

The `ImportPrivateKey` utility is used to load a private key into a private keystore file.

Syntax

```
$ java utils.ImportPrivateKey keystore storepass storetype keypass alias
certfile keyfile keyfilepass
```

Argument	Definition
keystore	The name of the keystore. A new keystore is created if one does not exist.
storepass	The password to open the keystore.
storetype	The type of keystore being used. The following options are available: • Demo Identity and Demo Trust—The demonstration identity and trust keystores located in the *WL_HOME*`\server\lib` directory and configured by default and the `cacerts` file in the *JAVA_HOME*`\jre\lib\security` directory. • Custom Identity and Java Standard Trust—An identity keystore you create and the trusted CAs defined in the `cacerts` file in the *JAVA_HOME*`\jre\lib\security` directory. • Custom Identity and Custom Trust—Identity and trust keystores you create. • Custom Identity and Command-Line Trust—An identity keystore you create and command-line arguments that specify the location of the trust keystore. Use this option in a production environment when the administration port is enabled and Managed Servers are started on the command line instead of by the Node Manager.
keypass	The password used to retrieve the private key file from the keystore.
alias	The name that is used to look up certificates and keys in the keystore.

Argument	Definition
certfile	The name of the certificate associated with the private key.
keyfile	The name of the file holding the protected private key.
keyfilepass	The password used to unlock the private key file and to protect the private key in the keystore

Example

Use the following steps to:

- Generate a certificate and private key using the CertGen utility
- Create a keystore and store a private key using the ImportPrivateKey utility

To generate a certificate:

Note: By default, the CertGen utility looks for the CertGenCA.der and CertGenCAKey.der files in the current directory, or in the *WL_HOME*/server/lib directory, as specified in the weblogic.home system property or the CLASSPATH.

Alternatively, you can specify CA files on the command line. If you want to use the default settings, there is no need to specify CA files on the command line.

1. Enter the following command to generate certificate files named testcert with private key files named testkey:

```
$ java utils.CertGen mykeypass testcert testkey


Creating Domestic Key Strength - 1024

Encoding
................................................................
................................................................
................................................................
Created Private Key files - testkey.der and testkey.pem
Encoding
................................................................
................................................................
................................................................
Created Certificate files - testcert.der and testcert.pem
```

```
.........................................................
```

2. Convert the certificate from DER format to PEM format.

```
$ java utils.der2pem CertGenCA.der Encoding
.........................................................
.........................................................
```

3. Concatenate the certificate and the Certificate Authority (CA).

```
$ type testcert.pem CertGenCA.pem >> newcerts.pem
```

4. Create a new keystore named `mykeystore` and load the private key located in the `testkey.pem` file.

```
$ java utils.ImportPrivateKey mykeystore mypasswd mykey
mykeypass newcerts.pem testkey.pem
Keystore file not found, creating it
```

jhtml2jsp

Converts JHTML files to JSP files. Be sure to inspect results carefully. Given the unpredictability of the JHTML code, `jhtml2jsp` will not necessarily produce flawless translations.

Output is a new JSP file named after the original file.

The HTTP servlets auto-generated from JSP pages differ from the regular HTTP servlets generated from JHTML. JSP servlets extend `weblogic.servlet.jsp.JspBase`, and so do not have access to the methods available to a regular HTTP servlet.

If your JHTML pages may reference these methods to access the servlet 'context' or 'config' objects, you will need to substitute these methods with the reserved words in JSP that represent these implicit objects.

If your JHTML uses variables that have the same name as the reserved words in JSP, the tool will output a warning. You will need to edit your Java code in the generated JSP page to change the variable name to something other than a reserved word.

Syntax

```
$ java weblogic.utils.jhtml2jsp -d <directory> filename.jhtml
```

or

```
$ java weblogic.utils.jhtml2jsp filename.jhtml
```

Argument	Definition
-d	Specify the target directory. If target directory isn't specified, output is written to current directory.

jspc

Deprecated JSP-specific compiler task. Use "appc" on page 3-3.

logToZip

The `logToZip` utility searches an HTTP server log file, 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 shown below, 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
```

MBean Commands

Use the MBean commands (CREATE, DELETE, GET, INVOKE, QUERY, and SET) to administer MBeans. See Commands for Managing WebLogic Server MBeans in *weblogic.Admin Command-Line Reference*.

MulticastTest

The `MulticastTest` utility helps you debug multicast problems when configuring a WebLogic Cluster. The utility sends out multicast packets and returns information about how effectively multicast is working on your network. Specifically, `MulticastTest` displays the following types of information via standard output:

1. A confirmation and sequence ID for each message sent out by the current server.
2. The sequence and sender ID of each message received from any clustered server, including the current 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 machine. For more information about configuring a cluster, see *Using WebLogic Server Clusters*.

Syntax

```
$ java utils.MulticastTest -n name -a address [-p portnumber]
   [-t timeout] [-s send] [-l timetolive]
```

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*	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 is 237.0.0.1.)
`-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.)
-i interfaceaddress	Optional. Address of interface card to use. Uses the default NIC if none is specified.
`-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` *sendpause*	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`.
-l timetolive	Optional. Sets how many hops a packet can make before being discarded. The default value is 1, which restricts the packet to the subnet.

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
................................................................
................................................................
................................................................
................................................................
................................................................
```

Pointbase and the Pointbase ant Task

PointBase is bundled with WebLogic Server as a sample database. It is documented at WL_HOME\common\eval\pointbase\docs, where WL_HOME is the WebLogic Server installation directory.

The WebLogic Server MedRec Pointbase Ant task starts up Pointbase and configures it to run with the MedRec application. The Pointbase Ant task creates and configures connection pools, datasources, connection factories, JMS tables, JMS servers, a mail session, and a security realm. Examine the task at WL_HOME/samples/server/medrec/setup/build.xml, where WL_HOME is your WebLogic Server installation directory. See the Medical Records Development Tutorials for more information.

PointBase documentation is also available at http://www.pointbase.com/support/docs/overview.html.

rmic

The WebLogic RMI compiler is a command-line utility for generating and compiling remote objects. Use `weblogic.rmic` to generate dynamic proxies on the client-side for custom remote object interfaces in your application, and to provide hot code generation for server-side objects. See WebLogic RMI Compiler in *Programming WebLogic RMI*.

Schema

The `Schema` utility lets you upload SQL statements to a database using the WebLogic JDBC drivers. For additional information about database connections, see *Programming WebLogic JDBC*.

Syntax

```
$ java utils.Schema driverURL driverClass [-u username]
   [-p password] [-verbose] SQLfile
```

Argument	Definition
driverURL	Required. URL for the JDBC driver.
driverClass	Required. Pathname of the JDBC driver class.
-u *username*	Optional. Valid username.
-p *password*	Optional. Valid password for the user.
-verbose	Optional. Prints SQL statements and database messages.
SQLfile	Required. Text file with SQL statements.

Example

The following code shows a `Schema` command line for the `examples.utils` package:

```
$ java utils.Schema
"jdbc:pointbase:server://localhost/demo"
"com.pointbase.jdbc.jdbcUniversalDriver" -u "examples"
-p "examples" examples/utils/ddl/demo.ddl
```

```
utils.Schema will use these parameters:
```

```
url: jdbc:pointbase:server://localhost/demo
driver: com.pointbase.jdbc.jdbcUniversalDriver
dbserver: null
user: examples
password: examples
SQL file: examples/utils/ddl/demo.ddl
```

servicegen

The `servicegen` Ant task takes as input an EJB JAR file or a list of Java classes, and creates all the needed Web Service components and packages them into a deployable EAR file. See `servicegen` in *Programming WebLogic Web Services*.

SearchAndBuild

This `Ant` task executes `build.xml` files that are included within the `FileSet`. The task assumes that all of the files defined in `FileSet` are valid build files, and executes the `Ant` task of each of them.

Make certain that your `FileSet` filtering is correct. If you include the `build.xml` file that `SearchAndBuildTask` is being called from, you will be stuck in an infinite loop as this task will execute the top level build file—itself—forever. See FileSet at `http://ant.apache.org/manual/CoreTypes/fileset.html`.

Example

```
<project name="all_modules" default="all" basedir=".">
<taskdef name="buildAll"
classname="weblogic.ant.taskdefs.build.SearchAndBuildTask"/>
<target name="all">
<buildAll>
<fileset dir="${basedir}">
<include name="**\build.xml"/>
<exclude name="build.xml"/>
</fileset>
</buildAll>
</target>
</project>
```

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
```

source2wsdd

Generates a `web-services.xml` deployment descriptor file from the Java source file for a Java class-implemented WebLogic Web Service. For complete documentation of this Ant task, see Running the source2wsdd Ant Task in *Programming WebLogic Web Services*.

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
```

ValidateCertChain

WebLogic Server provides the `ValidateCertChain` utility to check whether or not an existing certificate chain will be rejected by WebLogic Server. The utility uses certificate chains from PEM files, PKCS-12 files, PKCS-12 keystores, and JKS keystores. A complete certificate chain must be used with the utility. The following is the syntax for the `ValidateCertChain` utility:

```
java utils.ValidateCertChain -file pemcertificatefilenamejava
utils.ValidateCertChain -pem pemcertificatefilenamejava
utils.ValidateCertChain -pkcs12store pkcs12storefilenamejava
utils.ValidateCertChain -pkcs12file pkcs12filename passwordjava
utils.ValidateCertChain -jks alias storefilename [storePass]
```

Example of valid certificate chain:

```
java utils.ValidateCertChain -pem zippychain.pemCert[0]:
CN=zippy,OU=FOR
TESTINGONLY,O=MyOrganization,L=MyTown,ST=MyState,C=USCert[1]:
CN=CertGenCAB,OU=FOR
TESTINGONLY,O=MyOrganization,L=MyTown,ST=MyState,C=US

Certificate chain appears valid
```

Example of invalid certificate chain:

```
java utils.ValidateCertChain -jks mykey mykeystoreCert[0]:
CN=corba1,OU=FOR TESTING ONLY,
O=MyOrganization,L=MyTown,ST=MyState,C=US

CA cert not marked with critical BasicConstraint indicating it is a
CACert[1]: CN=CACERT,OU=FOR TESTING ONLY,
O=MyOrganization,L=MyTown,ST=MyState,C=USCertificate chain is invalid
```

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.

Example

```
$ java -verbose myapplication > & classList.tmp
$ java utils.verboseToZip classList.tmp app2.zip
```

wlappc

This Ant task invokes `appc`, a utility compiles and validates a J2EE EAR file, an EJB JAR file or a WAR file for deployment.

For more information, see“appc” at http://e-docs.bea.com/wls/docs81/ejb/appc_ejbc.html#appc.

wlcompile

Use the `wlcompile` Ant task to invoke the `javac` compiler to compile your application's Java files in a split development directory structure. See Developing Applications Using the Split Development Directory Structure in *Developing WebLogic Server Applications*.

wlconfig

The wlconfig Ant task enables you to configure a WebLogic Server domain by creating, querying, or modifying configuration MBeans on a running Administration Server instance. For complete documentation on this Ant task, see Configuring a WebLogic Server Domain Using the wlconfig Ant Task in the *WebLogic Server Command Reference*.

wldeploy

The `wldeploy` Ant task enables you to perform `Deployer` functions using attributes specified in an Ant task. See `wldeploy` Ant Task in *Deployment Tools Reference*.

wlpackage

You use the `wlpackage` Ant task to package your split development directory application as a traditional EAR file that can be deployed to WebLogic Server. See Developing Applications Using the Split Development Directory Structure in *Developing WebLogic Server Applications*.

wlserver

The `wlserver` Ant task enables you to start, reboot, shutdown, or connect to a WebLogic Server instance. The server instance may already exist in a configured WebLogic Server domain, or you can create a new single-server domain for development by using the `generateconfig=true` attribute. For complete documentation on this Ant task, see Starting Servers and Creating Domains Using the wlserver Ant Task in the *WebLogic Server Command Reference*.

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 -Dbea.home=path
```

Argument	Definition
`-nowrite`	Required. Sends the output to `stdout` instead of `writeLicense.txt`.
`-Dbea.home`	Required. Sets WebLogic system home (the root directory of your WebLogic Server installation).

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
```

wsdl2Service

The `wsdl2Service` Ant task is a Web Services tool that takes as input an existing WSDL file and generates the Java interface that represents the implementation of your Web Service and the web-services.xml file that describes the Web Service. See wsdl2Service in *Programming WebLogic Web Services*.

wsdlgen

The wsdlgen Ant task is a Web Services tool that generates a WSDL file from the EAR and WAR files that implement your Web Service. See wsdlgen in *Programming WebLogic Web Services*.

wspackage

Use the Web Services wspackage Ant task to package the various components of a WebLogic Web Service into a new deployable EAR file and add extra components to an already existing EAR file. See wspackage in *Programming WebLogic Web Services*.

weblogic.Server Command-Line Reference

The `weblogic.Server` class is the main class for a WebLogic Server instance. You start a server instance by invoking `weblogic.Server` in a Java command. You can invoke the class directly in a command prompt (shell), indirectly through scripts, or through the Node Manager.

This section describes the following:

- "Required Environment and Syntax for weblogic.Server" on page 4-1
- "Default Behavior" on page 4-3
- "weblogic.Server Configuration Options" on page 4-4
- "Using the weblogic.Server Command Line to Start a Server Instance" on page 4-27
- "Using the weblogic.Server Command Line to Create a Domain" on page 4-28
- "Verifying Attribute Values That Are Set on the Command Line" on page 4-29

For information about using scripts to start an instance of WebLogic Server, refer to "Starting Administration Servers" and "Starting Managed Servers From a WebLogic Server Script" in the Administration Console Online Help.

For information about using the Node Manager to start an instance of WebLogic Server, refer to "Overview of Node Manager" in the *Configuring and Managing WebLogic Server* guide.

Required Environment and Syntax for weblogic.Server

This section describes the environment that you must set up before you can start a server instance. Then it describes the syntax for invoking `weblogic.Server`.

Environment

To set up your environment for the `weblogic.Server` command:

1. Install and configure the WebLogic Server software, as described in the *WebLogic Server Installation Guide.*
2. Add WebLogic Server classes to the `CLASSPATH` environment variable, as described in "Modifying the Classpath" on page 4-2.
3. Include a Java Virtual Machine (JVM) in your `PATH` environment variable. You can use any JVM that is listed in the Supported Configurations page at http://e-docs.bea.com/wls/certifications/index.html.

   If you do not include a JVM in the `PATH` environment variable, you must provide a pathname for the Java executable file that the JVM provides.

Modifying the Classpath

WebLogic Server's classpath is already set, but you may choose to modify it for a number of reasons such as adding a patch to WebLogic Server, adding a third-party database, updating the version of the database you are using, or adding support for Log4j logging.

To apply a patch to ALL of your WebLogic Server domains without the need to modify the classpath of a domain, just give the patch JAR file the name, `weblogic_sp.jar`, and copy it into the *`WL_HOME`*`/server/lib` directory. The `commEnv.cmd/sh` script will automatically include a JAR named `weblogic_sp` on the classpath for you.

If you would rather not use the name `weblogic_sp.jar` for your patch file or you would just like to make sure a JAR file, such as one mentioned below, comes before `weblogic.jar` on the classpath:

- For ALL domains, edit the `commEnv.cmd/sh` script in *`WL_HOME`*`/common/bin` and prepend your JAR file to the `weblogic_classpath` environment variable.
- To apply a patch to a SPECIFIC WebLogic Server domain, edit the `setDomainEnv.cmd/sh` script, in that domain's bin directory, and prepend it onto the `PRE_CLASSPATH` environment variable.

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

```
WL_HOME/common/eval/pointbase/lib/
pbserver44.jar and pbclient44.jar
```

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

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

The shell environment in which you run a server determines which character you use to separate path elements. On Windows, you typically use a semicolon (;). In a BASH shell, you typically use a colon (:).

Syntax

The syntax for invoking `weblogic.Server` is as follows:

```
java [options] weblogic.Server [-help]
```

The `java weblogic.Server -help` command returns a list of frequently used options.

Default Behavior

If you have set up the required environment described in "Environment" on page 4-2, when you enter the command `java weblogic.Server` with no options, WebLogic Server does the following:

1. Looks in the current directory for a file named `config.xml`.
2. If `config.xml` exists in the current directory, WebLogic Server does the following:
   a. If only one server instance is defined in `./config.xml`, it starts that server instance.

      For example, if you issue `java weblogic.Server` from `WL_HOME\samples\domains\medrec`, WebLogic Server starts the MedRec server.

   b. If there are multiple server instances defined in `./config.xml`, WebLogic Server looks for a server configuration named `myserver`. If it finds such a server configuration, it starts the `myserver` instance.

      If it does not find a server named `myserver`, WebLogic Server exits the `weblogic.Server` process and generates an error message.
3. If there is no `config.xml` file in the current directory, WebLogic Server asks if you want to create a domain and server instance. If you answer yes, WebLogic Server does the following:
   a. Creates a server configuration named `myserver`, and persists the configuration in a file named `./config.xml`.

Any options that you specify are persisted to the `config.xml` file. For example, if you specify `-Dweblogic.ListenPort=8001`, then WebLogic Server saves `8001` in the config.xml file. For any options that you do not specify, the server instance uses default values.

WebLogic Server also creates a `config.xml.booted` file, which is a copy of the `config.xml` file in its state just after the server successfully boots. If the `config.xml` becomes corrupted, you can boot the server with this `config.xml.booted` file.

Note: For information about restrictions for creating and using domains, see Domain Restrictions in *Configuring and Managing WebLogic Server*.

b. Uses the username and password that you supply to create a user with administrative privileges. It stores the definition of this user along with other basic, security-related data in files named `DefaultAuthenticatorInit.ldift` and `SerializedSystemIni.dat`.

WebLogic Server also encrypts and stores your username and password in a `boot.properties` file, which enables you to bypass the login prompt during subsequent instantiations of the server. For more information, refer to "Boot Identity Files" in the Administration Console Online Help.

c. Creates two scripts, `startmydomain.cmd` and `startmydomain.sh`, that you can use to start subsequent instantiations of the server. You can use a text editor to modify startup options such as whether the server starts in production mode or development mode. The `startmydomain` script contains comments that describe each option.

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

weblogic.Server Configuration Options

You can use `weblogic.Server` options to configure the following attributes of a server instance:

- "JVM Parameters" on page 4-5
- "Location of License and Configuration Data" on page 4-6
- "Server Communication" on page 4-9
- "SSL" on page 4-14
- "Security" on page 4-17
- "Message Output and Logging" on page 4-22

- "Other Server Configuration Options" on page 4-24
- "Clusters" on page 4-27

Unless you are creating a new domain as described in "Using the weblogic.Server Command Line to Create a Domain" on page 4-28, all startup options apply to the current server instantiation; they do not modify the persisted values in an existing `config.xml` file. Use the Administration Console or the `weblogic.Admin` command to modify the `config.xml` file.

For information on verifying the WebLogic Server attribute values that you set, refer to "Verifying Attribute Values That Are Set on the Command Line" on page 4-29.

JVM Parameters

The following table describes frequently used options that configure the Java Virtual Machine (JVM) in which the server instance runs. For a complete list of JVM options, refer to the documentation for your specific JVM. For a list of JVMs that can be used with WebLogic Server, refer to the Supported Configurations page at http://e-docs.bea.com/wls/certifications/index.html.

Table 4-1 Frequently Used Options for Setting JVM Parameters

Option	Description
`-Xms` and `-Xmx`	Specify the minimum and maximum values (in megabytes) for Java heap memory. For example, you might want to start the server with a default allocation of 200 megabytes of Java heap memory to the WebLogic Server. To do so, you can start the server with the `java -Xms200m` and `-Xmx200m` options. For best performance it is recommended that the minimum and maximum values be the same so that the JVM does not resize the heap. The values assigned to these parameters can dramatically affect the performance of your WebLogic Server and are provided here only as general defaults. In a production environment you should carefully consider the correct memory heap size to use for your applications and environment.
`-classpath`	The minimum content for this option is described under "Modifying the Classpath" on page 4-2. Instead of using this argument, you can use an environment variable named `CLASSPATH` to specify the classpath.
`-client` `-server`	Used by some JVMs to start a HotSpot virtual machine, which enhances performance. For a list of JVMs that can be used with WebLogic Server, refer to the Supported Configurations page at http://e-docs.bea.com/wls/certifications/certifications/index.html.

Location of License and Configuration Data

All server instances must have access to license and configuration data. The following table provides options for indicating the location of this data.

Table 4-2 Options for Indicating the Location of License and Configuration Data

Option	Description
`-Dbea.home=`*`bea_home`*	Specifies the location of the BEA home directory, which contains licensing and other essential information. By default, `weblogic.Server` determines the location of the BEA home directory based on values in the classpath.
`-Dweblogic.RootDirectory=`*`path`*	Specifies the server's root directory. By default, the root directory is the directory from which you issue the start command. For more information, refer to "A Server's Root Directory" in the *Configuring and Managing WebLogic Server* guide.

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

Table 4-2 Options for Indicating the Location of License and Configuration Data

Option	Description
`-Dweblogic.management.GenerateDefaultConfig=true`	Prevents the `weblogic.Server` class from prompting for confirmation when creating a `config.xml` file. Valid only if you invoke `weblogic.Server` in an empty directory. See "Default Behavior" on page 4-3.
`-Dweblogic.Domain=`*`domain`*	Specifies the name of the domain. If you are using `weblogic.Server` to create a domain, you can use this option to give the domain a specific name. In addition, this option supports a directory structure that WebLogic Server required in releases prior to 7.0 and continues to support in current releases. Prior to 7.0, all configuration files were required to be located at the following pathname: `.../config/`*`domain`*`/config.xml` where *`domain`* is the name of the domain. If your domain's configuration file conforms to that pathname, and if you invoke the `weblogic.Server` command from a directory other than `config/`*`domain`*, you can include the `-Dweblogic.Domain=`*`domain`* argument to cause WebLogic Server to search for a `config.xml` file in a pathname that matches `config/`*`domain`*`/config.xml`.

For information on how a Managed Server retrieves its configuration data, refer to the `-Dweblogic.management.server` entry in Table 4-3 on page 11.

The Administration Console does not display values that you set on the command line. For information on verifying the attribute values that you set, refer to "Verifying Attribute Values That Are Set on the Command Line" on page 4-29.

Examples

The following example starts an Administration Server instance named SimpleServer. In the example, the `config.xml` file has been renamed to `SimpleDomain.xml` and it is located in a directory named `c:\my_domains\SimpleDomain`. The command itself is issued from the `D:\` directory after running *`WL_HOME`*`\server\bin\setWLSEnv.cmd` (where *`WL_HOME`* is the directory in which you installed WebLogic Server):

```
D:\> java -Dweblogic.Name=SimpleServer
-Dweblogic.ConfigFile=SimpleDomain.xml
-Dweblogic.RootDirectory=c:\my_domains\SimpleDomain weblogic.Server
```

The following example starts a Managed Server instance named SimpleManagedServer. Specifying a `config.xml` file is not valid because Managed Servers contact the Administration Server for their configuration data. Multiple instances of WebLogic Server can use the same root directory. However, if your server instances share a root directory, make sure that all relative filenames are unique. In this example, SimpleManagedServer shares its root directory with SimpleServer. The command itself is issued from the `D:\` directory after running *`WL_HOME`*`\server\bin\setWLSEnv.cmd`:

```
D:\> java -Dweblogic.Name=SimpleManagedServer
-Dweblogic.management.server=http://localhost:7001
-Dweblogic.RootDirectory=c:\my_domains\SimpleDomain weblogic.Server
```

Server Communication

The following table describes the options for configuring how servers communicate.

Table 4-3 Options for Configuring Server Communication

Option	Description

`-Dweblogic.management.server=`
`[`*`protocol`*`://]`*`Admin-host`*`:`*`port`*
	Starts a server instance as a Managed Server and specifies the Administration Server that will configure and manage the server instance.
	The domain's configuration file does not specify whether a server configuration is an Administration Server or a Managed Server. You determine whether a server instance is in the role of Administration Server or Managed Server with the options that you use to start the instance. If you omit the `-Dweblogic.management.server` option in the start command, the server starts as an Administration Server (although within a given domain, there can be only one active Administration Server instance). Once an Administration Server is running, you must start all other server configurations as Managed Servers by including the `-Dweblogic.management.server` option in the start command.
	For *`protocol`*, specify `HTTP`, `HTTPS`, `T3`, or `T3S`. The T3S and HTTPS protocols require you to enable SSL on the Managed Server and the Administration Server and specify the Administration Server's SSL listen port.
	Note: Regardless of which protocol you specify, the initial download of a Managed Server's configuration is over HTTP or HTTPS. After the RMI subsystem initializes, the server instance can use the T3 or T3S protocol.
	For *`Admin-host`*, specify `localhost` or the DNS name or IP address of the machine where the Administration Server is running.
	For *`port`*, specify the Administration Server's listen port. If you set up the domain-side administration port, *`port`* must specify the domain-wide administration port.
	For more information on configuring a connection to the Administration Server, refer to "Configuring a Connection to the Administration Server" in the Administration Console Online Help.

Table 4-3 Options for Configuring Server Communication

Option	Description
`-Dweblogic.ListenAddress=`*`host`*	Specifies the address at which this server instance listens for requests. The *`host`* value must be either the DNS name or the IP address of the computer that is hosting the server instance. This startup option overrides any listen address value specified in the `config.xml` file. The override applies to the current server instantiation; it does not modify the value in the `config.xml` file. Use the Administration Console or the `weblogic.Admin` command to modify the `config.xml` file. For more information, refer to "Configuring the Listen Address" in the *Configuring and Managing WebLogic Server* guide.

`-Dweblogic.ListenPort=`
`portnumber`
	Enables and specifies the plain-text (non-SSL) listen port for the server instance.
	This startup option overrides any listen port value specified in the `config.xml` file. The override applies to the current server instantiation; it does not modify the value in the `config.xml` file. Use the Administration Console or the `weblogic.Admin` command to modify the `config.xml` file.
	The default listen port is 7001.
	For more information, refer to "Configuring the Listen Ports" in the *Configuring and Managing WebLogic Server* guide.

Table 4-3 Options for Configuring Server Communication

Option	Description
`-Dweblogic.ssl.ListenPort=`*`portnumber`*	Enables and specifies the port at which this WebLogic Server instance listens for SSL connection requests. This startup option overrides any SSL listen port value specified in the `config.xml` file. The override applies to the current server instantiation; it does not modify the value in the `config.xml` file. Use the Administration Console or the `weblogic.Admin` command to modify the `config.xml` file. The default SSL listen port is 7002. For more information, refer to "Configuring the Listen Ports" in the *Configuring and Managing WebLogic Server* guide.
`-Dweblogic.management.discover={true | false}`	Determines whether an Administration Server recovers control of a domain after the server fails and is restarted. A `true` value causes an Administration Server to communicate with all knows Managed Servers and inform them that the Administration Server is running. A `false` value prevents an Administration Server from communicating with any Managed Servers that are currently active in the domain. **Caution:** Specify `false` for this option only in the development environment of a single server. Specifying `false` can cause server instances in the domain to have an inconsistent set of deployed modules. For information on re-establishing administrative control over Managed Servers after an Administration Server has already started, refer to "DISCOVERMANAGEDSERVER" on page 1-21.

The Administration Console does not display values that you set on the command line. For information on verifying the attribute values that you set, refer to "Verifying Attribute Values That Are Set on the Command Line" on page 4-29.

SSL

Each Weblogic Server instance uses an instance of `weblogic.management.security.SSLMBean` to represent its SSL configuration. All of the options in the following table that start with `-Dweblogic.security.SSL` modify the configuration of the server's `SSLMBean`. For example, the `-Dweblogic.security.SSL.ignoreHostnameVerification` option sets the value of the `SSLMBean`'s `ignoreHostnameVerification` attribute.

The following table describes the options for configuring a server to communicate using Secure Sockets Layer (SSL).

Table 4-4 Options for Configuring SSL

Option	Description

`-Dweblogic.security.SSL.`
`ignoreHostnameVerification=`
`true`
	Disables host-name verification, which enables you to use the demonstration digital certificates that are shipped with WebLogic Server.
	By default, when a WebLogic Server instance is in the role of SSL client (it is trying to connect to some other server or application via SSL), it verifies that the host name that the SSL server returns in its digital certificate matches the host name of the URL used to connect to the SSL server. If the host names do not match, the connection is dropped.
	If you disable host name verification, either by using this option or by modifying the server's configuration in the `config.xml` file, the server instance does not verify host names when it is in the role of SSL client.
	Note: BEA does not recommend using the demonstration digital certificates or turning off host name verification in a production environment.
	This startup option overrides any Host Name Verification setting in the `config.xml` file. The override applies to the current server instantiation; it does not modify the value in the `config.xml` file. Use the Administration Console or the `weblogic.Admin` command to modify the `config.xml` file.
	For more information, refer to "Using Hostname Verification" in the *Managing WebLogic Security* guide.

`-Dweblogic.security.SSL.`
`HostnameVerifier=`
`hostnameverifierimplmentation`
	Specifies the name of a custom Host Name Verifier class. The class must implement the `weblogic.security.SSL.HostnameVerifier` interface.

Table 4-4 Options for Configuring SSL

Option	Description

`-Dweblogic.security.SSL.`
`sessionCache.ttl=`
`sessionCacheTimeToLive`
	Modifies the default server-session time-to-live for the SSL session.
	The *`sessionCacheTimeToLive`* value specifies (in milliseconds) the time to live for the SSL session. The default value is 90000 milliseconds (90 seconds). This means if a client accesses the server again (via the same session ID) within 90 seconds, WebLogic Server will use the existing SSL session. You can change this value by setting -Dweblogic.security.SSL.sessionCache.ttl in the server startup script.
	For `sessionCache.ttl`:
	• The minimum value is `1`
	• The maximum value is `Integer.MAX_VALUE`
	• The default value is `90000`

`-Dweblogic.management.`
`pkpassword=`*`pkpassword`*
	Specifies the password for retrieving SSL private keys from an encrypted flat file.
	Use this option if you store private keys in an encrypted flat file.

`-Dweblogic.security.SSL.`
`trustedCAKeyStore=`*`path`*
	Deprecated.
	If you configure a server instance to use the SSL features that were available before WebLogic Server 8.1, you can use this argument to specify the certificate authorities that the server or client trusts. The *`path`* value must be a relative or qualified name to the Sun JKS keystore file (contains a repository of keys and certificates).
	If a server instance is using the SSL features that were available before 8.1, and if you do not specify this argument, the WebLogic Server or client trusts all of the certificates that are specified in *`JAVA_HOME`*`\jre\lib\security\cacerts`.
	We recommend that you do not use the demonstration certificate authorities in any type of production deployment.
	For more information, refer to "Configuring SSL" in the *Managing WebLogic Security* guide.

The Administration Console does not display values that you set on the command line. For information on verifying the attribute values that you set, refer to "Verifying Attribute Values That Are Set on the Command Line" on page 4-29.

Setting Additional SSL Attributes

To set additional SSL attributes from the startup command, do the following:

1. To determine which SSL attributes can be configured from startup options, view the WebLogic Server Javadoc for the `SSLMBean` and `ServerMBean`. The Javadoc also indicates valid values for each attribute.

   Each attribute that `SSLMBean` and `ServerMBean` expose as a setter method can be set by a startup option.

2. To set attributes in the `SSLMBean`, add the following option to the start command: `-Dweblogic.ssl.`*`attribute-name`*`=`*`value`*

   where *`attribute-name`* is the name of the MBean's setter method without the `set` prefix.

3. To set attributes in the `ServerMBean`, add the following option to the start command: `-Dweblogic.server.`*`attribute-name`*`=`*`value`*

   where *`attribute-name`* is the name of the MBean's setter method without the `set` prefix.

For example, the `SSLMBean` exposes its `Enabled` attribute with the following setter method:

```
setEnabled()
```

To enable SSL for a server instance named MedRecServer, use the following command when you start MedRecServer:

```
java -Dweblogic.Name=MedRecServer
     -Dweblogic.ssl.Enabled=true weblogic.Server
```

The Administration Console does not display values that you set on the command line. For information on verifying the attribute values that you set, refer to "Verifying Attribute Values That Are Set on the Command Line" on page 4-29.

Security

The following table describes the options for configuring general security parameters.

Table 4-5 Options for General Security Parameters

Option	Description
`-Dweblogic.management.username=`*`username`*	Specifies the username under which the server instance will run. The username must belong to a role that has permission to start a server. For information on roles and permissions, refer to "Security Roles" in the *Securing WebLogic Resources* guide. This option prevents a server instance from using any boot identity file and overrides other startup options that cause a server to use boot identity files. For more information, refer to "Boot Identity Files" in the Administration Console Online Help.
`-Dweblogic.management.password=`*`password`*	Specifies the user password. This option prevents a server instance from using any boot identity file and overrides other startup options that cause a server to use boot identity files. For more information, refer to "Boot Identity Files" in the Administration Console Online Help.
`-Dweblogic.system.StoreBootIdentity=true`	Creates a `boot.properties` file in the server's root directory. The file contains the username and an encrypted version of the password that was used to start the server. Do not specify this argument in a server's `ServerStartMBean` (Remote Startup tab in the Administration Console). For more information, refer to "Specifying User Credentials When Starting a Server with the Node Manager" in the Administration Console Online Help. BEA recommends that you do not add this argument to a startup script. Instead, use it only when you want to create a `boot.properties` file. For more information, refer to "Boot Identity Files" in the Administration Console Online Help.

Table 4-5 Options for General Security Parameters

Option	Description

`-Dweblogic.system.`
`BootIdentityFile=`*`filename`*
	Specifies a boot identity file that contains a username and password.
	The *`filename`* value must be the fully qualified pathname of a valid boot identity file. For example: `-Dweblogic.system.BootIdentityFile=C:\BEA\wlserver8.1\user_config\mydomain\myidentity.prop`
	If you do not specify a filename, a server instance or the `weblogic.Admin SHUTDOWN` and `FORCESHUTDOWN` commands use the `boot.properties` file in the server's root directory.
	If there is no boot identity file:
	• When starting a server, the server instance prompts you to enter a username and password.
	• When using the `weblogic.Admin SHUTDOWN` and `FORCESHUTDOWN` commands, you must use the `-username` and `-password` arguments to provide user credentials.

`-Dweblogic.system.`
`RemoveBootIdentity=true`
	Removes the boot identity file after a server starts.

`-Dweblogic.security.anonymous`
`UserName=`*`name`*
	Assigns a user ID to anonymous users. By default, all anonymous users are identified with the string `<anonymous>`.
	To emulate the security behavior of WebLogic Server 6.x, specify `guest` for the *`name`* value and create a user named `guest` in your security realm.
	For more information, refer to "Users and Groups" in the *Securing WebLogic Resources* guide.

Table 4-5 Options for General Security Parameters

Option	Description

`-Djava.security.manager`
`-Djava.security.policy[=]=`*`filename`*
	Standard J2EE options that enable the Java security manager and specify a filename (using a relative or fully-qualified pathname) that contains Java 2 security policies.
	To use the WebLogic Server sample policy file, specify *`WL_HOME`*`\server\lib\weblogic.policy`.
	Using `-Djava.security.policy==`*`filename`* causes the policy file to override any default security policy. A single equal sign (=) causes the policy file to be appended to an existing security policy.
	For more information, refer to "Using the Java Security Manager to Protect WebLogic Resources" in the *Programming WebLogic Security* guide.

`-Dweblogic.security.fullyDelegateAuthorization=true`	By default, roles and security policies cannot be set for an EJB or Web application through the Administration Console unless security constraints were defined in the deployment descriptor for the EJB or Web application. Use this option when starting WebLogic Server to override this problem. This startup option does not work with EJBs or EJB methods that use `<unchecked>` or `<restricted>` tags or Web applications that do not have a role-name specified in the `<auth-constraint>` tag.

Table 4-5 Options for General Security Parameters

Option	Description
`-Dweblogic.management.anonymousAdminLookupEnabled=true`	Enables you to retrieve an `MBeanHome` interface without specifying user credentials. The `MBeanHome` interface is part of the WebLogic Server JMX API. If you retrieve `MBeanHome` without specifying user credentials, the interface gives you read-only access to the value of any MBean attribute that is not explicitly marked as protected by the Weblogic Server MBean authorization process. This startup option overrides the Anonymous Admin Lookup Enabled setting on the Security →General tab in the Administration Console. By default, the `MBeanHome` API allows access to MBeans only for WebLogic users who are in one of the default security roles. For more information, refer to "Security Roles" in the *Securing WebLogic Resources* guide.
`-Dweblogic.security.identityAssertionTTL=`*`seconds`*	Configures the number of seconds that the Identity Assertion cache stores a Subject. When using an Identity Assertion provider (either for an X.509 certificate or some other type of token), Subjects are cached within the server. This greatly enhances performance for servlets and EJB methods with `<run-as>` tags as well as for other places where identity assertion is used but not cached (for example, signing and encrypting XML documents). There might be some cases where this caching violates the desired semantics. By default, Subjects remain in the cache for 300 seconds, which is also the maximum allowed value. Setting the value to `0` disables the cache. Setting a high value generally improves the performance of identity assertion, but makes the Identity Assertion provider less responsive to changes in the configured Authentication provider. For example, a change in the user's group will not be reflected until the Subject is flushed from the cache and recreated.

Table 4-5 Options for General Security Parameters

Option	Description
`-Dweblogic.security.audit.auditLogDir`	Specifies a new directory location for the `DefaultAuditRecorder.log` file. Although an Auditing provider is configured per security realm, each server writes auditing data to its own log file in the server directory. By default, all auditing information recorded by the WebLogic Auditing provider is saved in *WL_HOME*\yourdomain\yourserver\\`DefaultAuditRecorder.log` For more information, see "Configuring a WebLogic Auditing Provider" in *Managing WebLogic Security*.
`-Dweblogic.security.ldap.maxSize=<max bytes>`	Limits the size of the data file used by the embedded LDAP server. When the data file exceeds the specified size, WebLogic Server eliminates from the data file space occupied by deleted entries.
`-Dweblogic.security.ldap.changeLogThreshold=<number of entries>`	Limits the size of the change log file used by the embedded LDAP server. When the change log file exceeds the specified number of entries, WebLogic Server truncates the change log by removing all entries that have been sent to all managed servers.

The Administration Console does not display values that you set on the command line. For information on verifying the attribute values that you set, refer to "Verifying Attribute Values That Are Set on the Command Line" on page 4-29.

Message Output and Logging

The following table describes options for configuring a server instance's message output.

Table 4-6 Options for Configuring Message Output

Option	Description
`-Dweblogic.Stdout="filename"`	Redirects the server and JVM's standard output stream to a file. You can specify a pathname that is fully qualified or relative to the WebLogic Server root directory. For more information, refer to "Redirecting System.out and System.err to a File" in the Administration Console Online Help.
`-Dweblogic.Stderr="filename"`	Redirects the server and JVM's standard error stream to a file. You can specify a pathname that is fully qualified or relative to the WebLogic Server root directory. For more information, refer to "Redirecting System.out and System.err to a File" in the Administration Console Online Help.

`-Dweblogic.domain.ConfigurationAuditType=`
`{ none|log|audit|logaudit }`
	Specifies whether the Administration Server generates Audit Events and/or log messages when a user changes or invokes management operations on domain resources.
	See "Configuration Auditing" in the Administration Console Online Help.

The Administration Console does not display values that you set on the command line. For information on verifying the attribute values that you set, refer to "Verifying Attribute Values That Are Set on the Command Line" on page 4-29.

Setting Logging Attributes

Each Weblogic Server instance uses an instance of `weblogic.management.configuration.LogMBean` to represent the configuration of its logging services.

To set values for `LogMBean` attributes from the startup command, do the following:

1. To determine which log attributes can be configured from startup options, view the WebLogic Server Javadoc for the `LogMBean`. The Javadoc also indicates valid values for each attribute.

   Each attribute that the `LogMBean` exposes as a setter method can be set by a startup option.

2. Add the following option to the start command:
   `-Dweblogic.log.attribute-name=value`

where *attribute-name* is the name of the MBean’s setter method without the set prefix.

The LogMBean exposes its FileName attribute with the following setter method:

```
setFileName()
```

To specify the name of the MedRecServer instance’s local log file, use the following command when you start MedRecServer:

```
java -Dweblogic.Name=MedRecServer
     -Dweblogic.log.FileName="C:\logfiles\myServer.log"
     weblogic.Server
```

The Administration Console does not display values that you set on the command line. For information on verifying the attribute values that you set, refer to “Verifying Attribute Values That Are Set on the Command Line” on page 4-29.

Other Server Configuration Options

The following table describes options for configuring additional attributes of a server instance.

Table 4-7 Options for Configuring Server Attributes

Option	Description
-Dweblogic.Name= *servername*	Specifies the name of the server instance that you want to start. The specified value must refer to the name of a server that has been defined in the domain’s config.xml file.
-Dweblogic.ProductionModeEnabled= {true | false}	Determines whether all servers in a domain start in production mode. This option is applicable only when starting the Administration Server. All Managed Servers start in the same mode as the Administration Server. A true value prevents a WebLogic Server from automatically deploying and updating applications that are in the *domain_name*/applications directory. If you do not specify this option, the assumed value is false. For information on the differences between development mode and production mode, refer to "Differences Between Configuration Startup Modes."

Table 4-7 Options for Configuring Server Attributes

Option	Description

`-Dweblogic.management.`
`startupMode=STANDBY`
	Starts a server and places it in the `STANDBY` state. To use this startup argument, the domain must be configured to use the domain-wide administration port.
	For information about administration ports, refer to "Enabling the Domain-Wide Administration Port" in the *Configuring and Managing WebLogic Server* guide.
	This startup option overrides any startup mode setting in the `config.xml` file. The override applies to the current server instantiation; it does not modify the value in the `config.xml` file. Use the Administration Console or the `weblogic.Admin` command to modify the `config.xml` file.
	If you do not specify this value (either on the command line or in `config.xml`), the default is to start in the `RUNNING` state.

`-Dweblogic.apache.xerces.`
`maxentityrefs=`*`numerical-value`*
	Limits the number of entities in an XML document that the WebLogic XML parser resolves.
	If you do not specify this option, the XML parser that WebLogic Server installs resolves 10,000 entity references in an XML document, regardless of how many an XML document contains.

`-Dweblogic.jsp.windows.caseSe`
`nsitive=true`
	Causes the JSP compiler on Windows systems to preserve case when it creates output files names.
	See "Running JSPC on Windows Systems" in *Programming WebLogic JSP*.

Table 4-7 Options for Configuring Server Attributes

Option	Description
`-Dweblogic.net.http.URLStreamHandlerFactory=`*`classname`*	Used to override the default WebLogic Server HTTP stream handler. To use this option, write a class that implements the `java.net.URLStreamHandlerFactory` interface. In addition to implementing the `createURLStreamHandler("http")` method specified by the interface, the class must include a `main()` method that calls `java.net.URL.setURLStreamHandlerFactory()` with an instance of the class as its argument. Set this system property to the name of this class. For more information, see the javadocs for the `java.net.URL.URLStreamHandlerFactory` interface.
`-Dweblogic.servlet.optimisticSerialization=true`	When `optimistic-serialization` is turned on, WebLogic Server does not serialize-deserialize context and request attributes upon `getAttribute(name)` when the request is dispatched across servlet contexts. This means that you must make sure that the attributes common to Web applications are scoped to a common parent classloader (application scoped) or you must place them in the system classpath if the two Web applications do not belong to the same application. When `optimistic-serialization` is turned off (default value), WebLogic Server serialize-deserializes context and request attributes upon `getAttribute(name)` to avoid the possibility of ClassCastExceptions. The optimistic-serialization value can also be specified at domain level in the `WebAppContainerMBean`, which applies for all Web applications. The value in weblogic.xml, if specified, overrides the domain level value. The default value is false.

The Administration Console does not display values that you set on the command line. For information on verifying the attribute values that you set, refer to "Verifying Attribute Values That Are Set on the Command Line" on page 4-29.

Clusters

The following table describes options for configuring additional attributes of a cluster.

Table 4-8 Options for Configuring Cluster Attributes

Option	Description
`-Dweblogic.cluster.` `multicastAddress`	Determines the Multicast Address that clustered servers use to send and receive cluster-related communications. By default, a clustered server refers to the Multicast Address that is defined in the `config.xml` file. Use this option to override the value in `config.xml`. **Note:** The Administration Console does not display values that you set on the command line. For information on verifying the attribute values that you set, refer to "Verifying Attribute Values That Are Set on the Command Line" on page 4-29. Regardless of how you set the Multicast Address, all servers in a cluster must communicate at the same Multicast Address.

Using the weblogic.Server Command Line to Start a Server Instance

A simple way to start a server instance is as follows:

1. In a command shell, set up the required environment variables by running the following script:
   `WL_HOME``\server\bin\setWLSEnv.cmd` (on Windows)
   `WL_HOME``/server/bin/setWLSEnv.sh` (on UNIX)

   where *`WL_HOME`* is the directory in which you installed the WebLogic Server software.

2. In the command shell, change to the directory that contains your domain's `config.xml` file. For example, change to the *`WL_HOME`*`\samples\domains\medrec` directory.

3. To start an Administration Server, enter the following command:
   ```
   java -Dweblogic.Name=servername weblogic.Server
   ```

   where *`servername`* is the name of a server configuration that already exists in the `config.xml` file.

For example, enter the following command to start the MedRec server:

```
java -Dweblogic.Name=MedRecServer weblogic.Server
```

4. If the domain's Administration Server is already running, and if you have already defined a Managed Server in the `config.xml` file, you can start a Managed Server as follows:

```
java -Dweblogic.Name=managed-server-name
-Dweblogic.management.server=url-for-Administration-Server
weblogic.Server
```

For example, if you create a Managed Server named MedRecManagedServer in the MedRec domain, you can enter the following command:

```
java -Dweblogic.Name=MedRecManagedServer
-Dweblogic.management.server=localhost:7001
weblogic.Server
```

Using the weblogic.Server Command Line to Create a Domain

You can use `weblogic.Server` to create a domain that contains a single server instance. You cannot use `weblogic.Server` to add Managed Server instances to a domain, nor can you use `weblogic.Server` to modify an existing domain.

As described in "Default Behavior" on page 4-3, if `weblogic.Server` is unable to find a `config.xml` file, it offers to create the file. Any command option that you specify and that corresponds to an attribute that is persisted in the `config.xml` file will be persisted. For example, the `-Dweblogic.Name` and `-Dweblogic.Domain` options specify the name of a server configuration and the name of a domain. If `weblogic.Server` is unable to find a `config.xml` file, both of these values are persisted in `config.xml`. However, the `-Dweblogic.system.BootIdentityFile` option, which specifies a file that contains user credentials for starting a server instance, is not an attribute that the `config.xml` file persists.

To create and instantiate a simple example domain and server, do the following:

1. In a command shell, set up the required environment variables by running the following script:
   `WL_HOME``\server\bin\setWLSEnv.cmd` (on Windows)
   `WL_HOME``/server/bin/setWLSEnv.sh` (on UNIX)

   where *`WL_HOME`* is the directory in which you installed the WebLogic Server software.

2. In the command shell, create an empty directory.

3. In the empty directory, enter the following command:

```
java -Dweblogic.Domain=SimpleDomain -Dweblogic.Name=SimpleServer
-Dweblogic.management.username=weblogic
-Dweblogic.management.password=weblogic   -Dweblogic.ListenPort=7701
weblogic.Server
```

After you enter this command, WebLogic Server asks if you want to create a new `config.xml` file. If you enter `y`, it asks you to confirm the password. Then it instantiates a domain named SimpleDomain. The domain's Administration Server is configured as follows:

- The name of the Administration Server is SimpleServer.
- The domain's security realm defines one administrative user, `weblogic`, with a password of `weblogic`.
- For the listen address of the Administration Server, you can use `localhost`, the IP address of the host computer, or the DNS name of the host computer. For more information about setting the listen address, refer to "Configuring the Listen Address" in the Administration Console Online Help.
- The Administration Server listens on port 7701.

Entering the `weblogic.Server` command as described in this section creates the following files:

- `config.xml`
- `DefaultAuthenticatorInit.ldift` and `SerializedSystemIni.dat`, which store basic security-related data.
- `boot.properties` file, which contains the username and password in an encrypted format. This file enables you to bypass the prompt for username and password when you start the server. For more information, refer to "Boot Identity Files" in the Administration Console Online Help.
- `startmydomain.cmd` and `startmydomain.sh`, that you can use to start subsequent instantiations of the server.

Verifying Attribute Values That Are Set on the Command Line

The Administration Console does not display values that you set on the command line because the startup options set attribute values for the server's Local Configuration MBean. To see the values that are in a server's Local Configuration MBean, use the `weblogic.Admin` utility as follows:

```
java weblogic.Admin -url url-for-server-instance -username username
-password password GET -type MBean-nameConfig -property attribute-name
```

For example, to determine the multicast address that a cluster member is using, enter the following command, where `MRMachine1:7041` is the listen address and port of the cluster member:

```
java weblogic.Admin -url MRMachine1:7041 -username weblogic -password
weblogic GET -pretty -type ClusterConfig -property MulticastAddress
```

To determine the severity level of messages that the example MedRecServer prints to standard out, enter the following command:

```
java weblogic.Admin -url localhost:7001 -username weblogic -password
weblogic GET -pretty -type ServerConfig -property StdoutSeverityLevel
```

For more information about Local Configuration MBeans, refer to "Overview of WebLogic JMX Services" in the *Programming WebLogic Server Management Services with JMX* guide. For more information on using the `weblogic.Admin` utility, refer to Chapter 1, “weblogic.Admin Command-Line Reference.”

Using Ant Tasks to Configure a WebLogic Server Domain

The following sections describe how to start and stop WebLogic Server instances and configure WebLogic Server domains using WebLogic Ant tasks:

- "Overview of Configuring and Starting Domains Using Ant Tasks" on page 5-1
- "Starting Servers and Creating Domains Using the wlserver Ant Task" on page 5-2
- "Configuring a WebLogic Server Domain Using the wlconfig Ant Task" on page 5-6

For information about restrictions for creating and using domains, see Domain Restrictions in *Configuring and Managing WebLogic Server*.

Overview of Configuring and Starting Domains Using Ant Tasks

WebLogic Server provides a pair of Ant tasks to help you perform common configuration tasks in a development environment. The configuration tasks enable you to start and stop WebLogic Server instances as well as create and configure WebLogic Server domains.

When combined with other WebLogic Ant tasks, you can create powerful build scripts for demonstrating or testing your application with custom domains. For example, a single Ant build script can:

- Compile your application using the `wlcompile`, `wlappc`, and Web Services Ant tasks.
- Create a new single-server domain and start the Administration Server using the `wlserver` Ant task.

- Configure the new domain with required application resources using the `wlconfig` Ant task.
- Deploy the application using the `wldeploy` Ant task.
- Automatically start a compiled client application to demonstrate or test product features.

The sections that follow describe how to use the configuration Ant tasks, `wlserver` and `wlconfig`. For more information about other Ant tasks included with WebLogic Server, see the *WebLogic Server Tools Reference*.

Starting Servers and Creating Domains Using the wlserver Ant Task

What the wlserver Ant Task Does

The `wlserver` Ant task enables you to start, reboot, shutdown, or connect to a WebLogic Server instance. The server instance may already exist in a configured WebLogic Server domain, or you can create a new single-server domain for development by using the `generateconfig=true` attribute.

When you use the `wlserver` task in an Ant script, the task does not return control until the specified server is available and listening for connections. If you start up a server instance using wlserver, the server process automatically terminates after the Ant VM terminates. If you only connect to a currently-running server using the `wlserver` task, the server process keeps running after Ant completes.

Basic Steps for Using wlserver

To use the wlserver Ant task:

1. Set your environment.

   On Windows NT, execute the `setWLSEnv.cmd` command, located in the directory `WL_HOME\server\bin`, where `WL_HOME` is the top-level directory of your WebLogic Server installation.

   On UNIX, execute the `setWLSEnv.sh` command, located in the directory `WL_HOME/server/bin`, where `WL_HOME` is the top-level directory of your WebLogic Server installation.

Note: The `wlserver` task is predefined in the version of Ant shipped with WebLogic Server. If you want to use the task with your own Ant installation, add the following task definition in your build file:

```
<taskdef name="wlserver"
classname="weblogic.ant.taskdefs.management.WLServer"/>
```

2. Add a call to the `wlserver` task in the build script to start, shutdown, restart, or connect to a server. See "wlserver Ant Task Reference" on page 5-4 for information about `wlserver` attributes and default behavior.

3. Execute the Ant task or tasks specified in the `build.xml` file by typing `ant` in the staging directory, optionally passing the command a target argument:

   ```
   prompt> ant
   ```

   Use `ant -verbose` to obtain more detailed messages from the `wlserver` task.

Sample build.xml Files for wlserver

The following shows a minimal `wlserver` target that starts a server in the current directory using all default values:

```
  <target name="wlserver-default">
    <wlserver/>
  </target>
```

This target connects to an existing, running server using the indicated connection parameters and username/password combination:

```
  <target name="connect-server">
    <wlserver host="127.0.0.1" port="7001" username="weblogic"
password="weblogic" action="connect"/>
  </target>
```

This target starts a WebLogic Server instance configured in the `config` subdirectory:

```
  <target name="start-server">
    <wlserver dir="./config" host="127.0.0.1" port="7001" action="start"/>
  </target>
```

This target creates a new single-server domain in an empty directory, and starts the domain's server instance:

```
  <target name="new-server">
    <delete dir="./tmp"/>
```

```
    <mkdir dir="./tmp"/>
    <wlserver dir="./tmp" host="127.0.0.1" port="7001"
    generateConfig="true" username="weblogic" password="weblogic"
action="start"/>
  </target>
```

wlserver Ant Task Reference

The following table describes the attributes of the `wlserver` Ant task.

Table 5-1 Attributes of the wlserver Ant Task

Attribute	Description	Data Type	Required?
policy	The path to the security policy file for the WebLogic Server domain. This attribute is used only for starting server instances.	File	No
dir	The path that holds the domain configuration (for example, `c:\bea\user_projects\mydomain`). By default, `wlserver` uses the current directory.	File	No
beahome	The path to the BEA home directory (for example, `c:\bea`).	File	No
weblogichome	The path to the WebLogic Server installation directory (for example, `c:\bea\weblogic81`).	File	No
servername	The name of the server to start, reboot, or connect to.	String	No
domainname	The name of the WebLogic Server domain in which the server is configured.	String	No
adminserverurl	The URL to access the Administration Server in the domain. This attribute is required if you are starting up a Managed Server in the domain.	String	Required for starting Managed Servers.
username	The username of an administrator account. If you omit both the `username` and `password` attributes, `wlserver` attempts to obtain the encrypted username and password values from the `boot.properties` file. See "Default Behavior" on page 4-3 for more information on `boot.properties`.	String	No

Table 5-1 Attributes of the wlserver Ant Task

Attribute	Description	Data Type	Required?
password	The password of an administrator account. If you omit both the `username` and `password` attributes, `wlserver` attempts to obtain the encrypted username and password values from the `boot.properties` file. See “Default Behavior” on page 4-3 for more information on `boot.properties`.	String	No
pkpassword	The private key password for decrypting the SSL private key file.	String	No
timeout	The maximum time, in seconds, that `wlserver` waits for a server to boot. This also specifies the maximum amount of time to wait when connecting to a running server.	long	No
productionmod eenabled	Specifies whether a server instance boots in development mode or in production mode.	boolean	No
host	The DNS name or IP address on which the server instance is listening.	String	No
port	The TCP port number on which the server instance is listening.	int	No
generateconfig	Specifies whether or not `wlserver` creates a new domain for the specified server. This attribute is false by default.	boolean	No
action	Specifies the action `wlserver` performs: `startup`, `shutdown`, `reboot`, or `connect`. The `shutdown` action can be used with the optional `forceshutdown` attribute perform a forced shutdown.	String	No

Table 5-1 Attributes of the wlserver Ant Task

Attribute	Description	Data Type	Required?
failonerror	This is a global attribute used by WebLogic Server Ant tasks. It specifies whether the task should fail if it encounters an error during the build. This attribute is set to true by default.	Boolean	No

forceshutdown	This optional attribute is used in conjunction with the `action="shutdown"` attribute to perform a forced shutdown. For example:
	`<wlserver`
	`  host="${wls.host}"`
	`  port="${port}"`
	`  username="${wls.username}"`
	`  password="${wls.password}"`
	`  action="shutdown"`
	`  forceshutdown="true"/>`
		Boolean	No

Configuring a WebLogic Server Domain Using the wlconfig Ant Task

What the wlconfig Ant Task Does

The `wlconfig` Ant task enables you to configure a WebLogic Server domain by creating, querying, or modifying configuration MBeans on a running Administration Server instance. Specifically, `wlconfig` enables you to:

- Create new MBeans, optionally storing the new MBean Object Names in Ant properties.
- Set attribute values on a named MBean available on the Administration Server.
- Create MBeans and set their attributes in one step by nesting set attribute commands within create MBean commands.
- Query MBeans, optionally storing the query results in an Ant property reference.
- Query MBeans and set attribute values on all matching results.
- Establish a parent/child relationship among MBeans by nesting create commands within other create commands.

Basic Steps for Using wlconfig

1. Set your environment in a command shell. See "Basic Steps for Using wlserver" on page 5-2 for details.

   Note: The `wlconfig` task is predefined in the version of Ant shipped with WebLogic Server. If you want to use the task with your own Ant installation, add the following task definition in your build file:

   ```
   <taskdef name="wlconfig"
   classname="weblogic.ant.taskdefs.management.WLConfig"/>
   ```

2. `wlconfig` is commonly used in combination with `wlserver` to configure a new WebLogic Server domain created in the context of an Ant task. If you will be using `wlconfig` to configure such a domain, first use `wlserver` attributes to create a new domain and start the WebLogic Server instance.

3. Add an initial call to the `wlconfig` task to connect to the Administration Server for a domain. For example:

   ```
   <target name=”doconfig”>
      <wlconfig url="t3://localhost:7001" username="weblogic"
         password="weblogic">
   </target>
   ```

4. Add nested `create`, `delete`, `get`, `set`, and `query` elements to configure the domain.

5. Execute the Ant task or tasks specified in the `build.xml` file by typing `ant` in the staging directory, optionally passing the command a target argument:

   ```
   prompt> ant doconfig
   ```

   Use `ant -verbose` to obtain more detailed messages from the `wlconfig` task.

Sample build.xml Files for wlconfig

Complete Example

This example shows a single `build.xml` file that creates a new domain using `wlserver` and performs various domain configuration tasks with `wlconfig`. The configuration tasks set up domain resources required by the Avitek Medical Records sample application.

The script starts by creating the new domain:

```
<target name="medrec.config">
   <mkdir dir="config"/>
   <wlserver username="a" password="a" servername="MedRecServer"
      domainname="medrec" dir="config" host="localhost" port="7000"
      generateconfig="true"/>
```

The script then starts the `wlconfig` task by accessing the newly-created server:

```
<wlconfig url="t3://localhost:7000" username="a" password="a">
```

Within the `wlconfig` task, the `query` element runs a query to obtain the Server MBean object name, and stores this MBean in the `${medrecserver}` Ant property:

```
 <query domain="medrec" type="Server" name="MedRecServer"
      property="medrecserver"/>
```

The script the uses a `create` element to create a new JDBC connection pool in the domain, storing the object name in the `${medrecpool}` Ant property. Nested `set` elements in the `create` operation set attributes on the newly-created MBean. The new pool is target to the server using the `${medrecserver}` Ant property set in the query above:

```
<create type="JDBCConnectionPool" name="MedRecPool"
   property="medrecpool">
   <set attribute="CapacityIncrement" value="1"/>
   <set attribute="DriverName"
      value="com.pointbase.jdbc.jdbcUniversalDriver"/>
   <set attribute="InitialCapacity" value="1"/>
   <set attribute="MaxCapacity" value="10"/>
   <set attribute="Password" value="MedRec"/>
   <set attribute="Properties" value="user=MedRec"/>
   <set attribute="RefreshMinutes" value="0"/>
   <set attribute="ShrinkPeriodMinutes" value="15"/>
   <set attribute="ShrinkingEnabled" value="true"/>
   <set attribute="TestConnectionsOnRelease" value="false"/>
   <set attribute="TestConnectionsOnReserve" value="false"/>
   <set attribute="URL"
      value="jdbc:pointbase:server://localhost/demo"/>
   <set attribute="Targets" value="${medrecserver}"/>
</create>
```

Next, the script creates a JDBC TX DataSource using the JDBC connection pool created above:

```
<create type="JDBCTxDataSource" name="Medical Records Tx DataSource">
   <set attribute="JNDIName" value="MedRecTxDataSource"/>
   <set attribute="PoolName" value="MedRecPool"/>
   <set attribute="Targets" value="${medrecserver}"/>
</create>
```

The script creates a new JMS connection factory using nested `set` elements:

```
<create type="JMSConnectionFactory" name="Queue">
   <set attribute="JNDIName" value="jms/QueueConnectionFactory"/>
   <set attribute="XAServerEnabled" value="true"/>
   <set attribute="Targets" value="${medrecserver}"/>
</create>
```

A new JMS JDBC store is created using the `MedRecPool`:

```
<create type="JMSJDBCStore" name="MedRecJDBCStore"
   property="medrecjdbcstore">
   <set attribute="ConnectionPool" value="${medrecpool}"/>
   <set attribute="PrefixName" value="MedRec"/>
</create>
```

When creating a new JMS server, the script uses a nested `create` element to create a JMS queue, which is the child of the JMS server:

```
<create type="JMSServer" name="MedRecJMSServer">
   <set attribute="Store" value="${medrecjdbcstore}"/>
   <set attribute="Targets" value="${medrecserver}"/>
   <create type="JMSQueue" name="Registration Queue">
      <set attribute="JNDIName" value="jms/REGISTRATION_MDB_QUEUE"/>
   </create>
</create>
```

This script creates a new mail session and startup class:

```
<create type="MailSession" name="Medical Records Mail Session">
   <set attribute="JNDIName" value="mail/MedRecMailSession"/>
   <set attribute="Properties"
      value="mail.user=joe;mail.host=mail.mycompany.com"/>
   <set attribute="Targets" value="${medrecserver}"/>
</create>
```

```
   <create type="StartupClass" name="StartBrowser">
      <set attribute="Arguments" value="port=${listenport}"/>
      <set attribute="ClassName"
         value="com.bea.medrec.startup.StartBrowser"/>
      <set attribute="FailureIsFatal" value="false"/>
      <set attribute="Notes" value="Automatically starts a browser on
server boot."/>

      <set attribute="Targets" value="${medrecserver}"/>

    </create>
```

Finally, the script obtains the WebServer MBean and sets the log filename using a nested `set` element:

```
      <query domain="medrec" type="WebServer" name="MedRecServer">
         <set attribute="LogFileName" value="logs/access.log"/>
      </query>
   </wlconfig>
</target>
```

Query and Delete Example

The `query` element does not need to specify an MBean name when nested within a `query` element:

```
<target name="queryDelete">
   <wlconfig url="${adminurl}" username="${user}" password="${pass}"
      failonerror="false">
      <query query="${wlsdomain}:Name=MyNewServer2,*"
         property="deleteQuery">
         <delete/>
      </query>
   </wlconfig>
</target>
```

Example of Setting Multiple Attribute Values

The `set` element allows you to set an attribute value to multiple object names stored in Ant properties. For example, the following target stores the object names of two servers in separate Ant properties, then uses those properties to assign both servers to the target attribute of a new JDBC Connection Pool:

```
<target name="multipleJDBCTargets">
   <wlconfig url="${adminurl}" username="${user}" password="${pass}">
      <query domain="mydomain" type="Server" name="MyServer"
         property="myserver"/>
      <query domain="mydomain" type="Server" name="OtherServer"
         property="otherserver"/>
      <create type="JDBCConnectionPool" name="sqlpool" property="sqlpool">
         <set attribute="CapacityIncrement" value="1"/>
[.....]
         <set attribute="Targets" value="${myserver};${otherserver}"/>
      </create>
   </wlconfig>
</target>
```

wlconfig Ant Task Reference

Main Attributes

The following table describes the main attributes of the `wlconfig` Ant task.

Table 5-2 Main Attributes of the wlconfig Ant Task

Attribute	Description	Data Type	Required?
url	The URL of the domain's Administration Server.	String	Yes
username	The username of an administrator account.	String	No

Table 5-2 Main Attributes of the wlconfig Ant Task

Attribute	Description	Data Type	Required?
password	The password of an administrator account. To avoid having the plain text password appear in the build file or in process utilities such as `ps`, first store a valid username and encrypted password in a configuration file using the `weblogic.Admin STOREUSERCONFIG` command. Then omit both the `username` and `password` attributes in your Ant build file. When the attributes are omitted, `wlconfig` attempts to login using values obtained from the default configuration file. If you want to obtain a username and password from a non-default configuration file and key file, use the `userconfigfile` and `userkeyfile` attributes with `wlconfig`. See "STOREUSERCONFIG" on page 1-15 for more information on storing and encrypting passwords.	String	No
failonerror	This is a global attribute used by WebLogic Server Ant tasks. It specifies whether the task should fail if it encounters an error during the build. This attribute is set to true by default.	Boolean	No

Table 5-2 Main Attributes of the wlconfig Ant Task

Attribute	Description	Data Type	Required?
userconfigfile	Specifies the location of a user configuration file to use for obtaining the administrative username and password. Use this option, instead of the `username` and `password` attributes, in your build file when you do not want to have the plain text password shown in-line or in process-level utilities such as `ps`. Before specifying the `userconfigfile` attribute, you must first generate the file using the `weblogic.Admin STOREUSERCONFIG` command as described in "STOREUSERCONFIG" on page 1-15.	File	No
userkeyfile	Specifies the location of a user key file to use for encrypting and decrypting the username and password information stored in a user configuration file (the `userconfigfile` attribute). Before specifying the `userkeyfile` attribute, you must first generate the key file using the `weblogic.Admin STOREUSERCONFIG` command as described in "STOREUSERCONFIG" on page 1-15.	File	No

Nested Elements

`wlconfig` also has several elements that can be nested to specify configuration options:

- create
- delete
- set
- get
- query

create

The `create` element creates a new MBean in the WebLogic Server domain. The `wlconfig` task can have any number of `create` elements.

A create element can have any number of nested set elements, which set attributes on the newly-created MBean. A create element may also have additional, nested create elements that create child MBeans.

The create element has the following attributes.

Table 5-3 Attributes of the create Element

Attribute	Description	Data Type	Required?
name	The name of the new MBean object to create.	String	No (wlconfig supplies a default name if none is specified.)
type	The MBean type.	String	Yes
property	The name of an optional Ant property that holds the object name of the newly-created MBean. **Note:** If you nest a create element inside of another create element, you cannot specify the property attribute for the *nested* create element.	String	No

delete

The `delete` element removes an existing MBean from the WebLogic Server domain. `delete` takes a single attribute:

Table 5-4 Attribute of the delete Element

Attribute	Description	Data Type	Required?
mbean	The object name of the MBean to delete.	String	Required when the `delete` element is a direct child of the `wlconfig` task. Not required when nested within a `query` element.

set

The `set` element sets MBean attributes on a named MBean, a newly-created MBean, or on MBeans retrieved as part of a query. You can include the `set` element as a direct child of the `wlconfig` task, or nested within a `create` or `query` element.

The `set` element has the following attributes:

Table 5-5 Attributes of the set Element

Attribute	Description	Data Type	Required?
attribute	The name of the MBean attribute to set.	String	Yes
value	The value to set for the specified MBean attribute. You can specify multiple object names (stored in Ant properties) as a value by delimiting the entire value list with quotes and separating the object names with a semicolon. See "Example of Setting Multiple Attribute Values" on page 5-10.	String	Yes

Table 5-5 Attributes of the set Element

Attribute	Description	Data Type	Required?
mbean	The object name of the MBean whose values are being set. This attribute is required only when the `set` element is included as a direct child of the main `wlconfig` task; it is not required when the set element is nested within the context of a `create` or `query` element.	String	Required only when the `set` element is a direct child of the `wlconfig` task.
domain	This attribute specifies the JMX domain name for Security MBeans and third-party SPI MBeans. It is not required for administration MBeans, as the domain corresponds to the WebLogic Server domain. **Note:** You cannot use this attribute if the `set` element is nested inside of a `create` element.	String	No

get

The `get` element retrieves attribute values from an MBean in the WebLogic Server domain. The `wlconfig` task can have any number of `get` elements.

The `get` element has the following attributes.

Table 5-6 Attributes of the get Element

Attribute	Description	Data Type	Required?
attribute	The name of the MBean attribute whose value you want to retrieve.	String	Yes

Table 5-6 Attributes of the get Element

Attribute	Description	Data Type	Required?
property	The name of an Ant property that will hold the retrieved MBean attribute value.	String	Yes
mbean	The object name of the MBean you want to retrieve attribute values from.	String	Yes

query

The `query` elements finds MBean that match a search pattern. `query` can be used with nested `set` elements or a nested `delete` element to perform set or delete operations on all MBeans in the result set.

`wlconfig` can have any number of nested `query` elements.

`query` has the following attributes:

Table 5-7 Attributes of the query Element

Attribute	Description	Data Type	Required?
domain	The name of the WebLogic Server domain in which to search for MBeans.	String	No
type	The type of MBean to query.	String	No
name	The name of the MBean to query.	String	No
pattern	A JMX query pattern.	String	No
property	The name of an optional Ant property that will store the query results.	String	No
domain	This attribute specifies the JMX domain name for Security MBeans and third-party SPI MBeans. It is not required for administration MBeans, as the domain corresponds to the WebLogic Server domain.	String	No

WebLogic SNMP Agent Command-Line Reference

WebLogic Server can use Simple Network Management Protocol (SNMP) to communicate with enterprise-wide management systems. The WebLogic Server subsystem that gathers WebLogic management data, converts it to SNMP communication modules (trap notifications), and forwards the trap notifications to third-party SNMP management systems is called the WebLogic SNMP agent. The WebLogic SNMP agent runs on the Administration Server and collects information from all Managed Servers within a domain.

The WebLogic SNMP agent provides a command-line interface that lets you:

- Retrieve the value of WebLogic Server attributes that are exposed as managed objects in the WebLogic Server Management Information Base (MIB).
- Generate and receive WebLogic Server traps.

The following sections describe working with the WebLogic SNMP agent through its command-line interface:

- "Required Environment and Syntax for the SNMP Command-Line Interface" on page 6-2
- "Commands for Retrieving the Value of WebLogic Server Attributes" on page 6-4
- "Commands for Testing Traps" on page 6-9

For more information about using SNMP with WebLogic Server, refer to the WebLogic SNMP Management Guide.

Required Environment and Syntax for the SNMP Command-Line Interface

Before you use the WebLogic SNMP agent command-line interface, set up your environment and note the command syntax information described in the following sections.

Environment

To set up your environment for the WebLogic SNMP agent command-line interface:

1. Install and configure the WebLogic Server software, as described in the *WebLogic Server Installation Guide.*
2. Enable the WebLogic SNMP agent, as described in "Configuring SNMP and WebLogic Server" in the *Administration Console Online Help*.

   Note: The `snmpv1trap` and `snmptrapd` commands do not require the SNMP agent to be enabled.

3. Open a command prompt (shell) and invoke the following script:

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

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

   The script adds a supported JDK to the shell's `PATH` environment variable and adds WebLogic Server classes to the `CLASSPATH` variable.

Common Arguments

All WebLogic SNMP agent commands take the following form:

```
java command-name arguments
```

Table 6-1 describes arguments that are common to most WebLogic SNMP agent commands.

Table 6-1 Common Command Line Arguments

Argument	Definition
`-d`	Includes debugging information and packet dumps in the command output.
`-v {v1 | v2}`	Specifies whether to use SNMPv1 or SNMPv2 to communicate with the SNMP agent. You must specify the same SNMP version that you set in the Trap Version field when you configured the SNMP agent (as described in "Enabling and Configuring the WebLogic SNMP Agent" in the *Administration Console Online Help*) If you do not specify a value, the command assumes `-v v1`.
`-c` *`snmpCommunity`* `[@`*`server_name`*` | @`*`domain_name`*`]`	Specifies the community name that the WebLogic SNMP agent uses to secure SNMP data **and** specifies the server instance that hosts the objects with which you want to interact. To request the value of an object on the Administration Server, specify: *`snmpCommunity`* where *`snmpCommunity`* is the SNMP community name that you set in the Community Prefix field when you configured the SNMP agent (as described in "Enabling and Configuring the WebLogic SNMP Agent" in the *Administration Console Online Help*). To request the value of an object on a Managed Server, specify: *`snmpCommunity@server_name`* where *`server_name`* is the name of the Managed Server. To request the value of an object for all server instances in a domain, specify a community string with the following form: *`snmpCommunity@domain_name`* If you do not specify a value, the command assumes `-c public`, which attempts to retrieve the value of an object that is on the Administration Server.
`-p` *`snmpPort`*	Specifies the port number on which the WebLogic SNMP agent listens for requests. If you do not specify a value, the command assumes `-p 161`.
`-t` *`timeout`*	Specifies the number of milliseconds the command waits to successfully connect to the SNMP agent. If you do not specify a value, the command assumes `-t 5000`.

Table 6-1 Common Command Line Arguments

Argument	Definition
`-r retries`	Specifies the number of times the command retries unsuccessful attempts to connect to the SNMP agent. If you do not specify a value, the command exits on the first unsuccessful attempt.
`host`	Specifies the DNS name or IP address of the computer that hosts the WebLogic Server Administration Server, which is where the WebLogic SNMP agent runs.

Commands for Retrieving the Value of WebLogic Server Attributes

Table 6-2 is an overview of commands that retrieve the value of WebLogic Server MBean attributes that are exposed in the WebLogic Server MIB.

Table 6-2 Overview of Commands for Retrieving the Value of WebLogic Server Attributes

Command	Description
`snmpwalk`	Returns a recursive list of all managed objects that are below a specified node in the MIB tree. See "snmpwalk" on page 6-4.
`snmpgetnext`	Returns a description of the managed object that immediately follows an OID that you specify. See "snmpgetnext" on page 6-6.
`snmpget`	Returns a description of managed objects that correspond to one or more object-instance OIDs. See "snmpget" on page 6-8.

snmpwalk

Returns a recursive list of all managed objects that are below a specified node in the MIB tree.

If you specify the OID for an object type, the command returns a list of all instances of that type along with all instances of any child object types.

For example, if you specify the OID for an object type that corresponds to an MBean, this command returns a description of all instances of the MBean **and** all instances of the attributes within the MBeans.

To see the WebLogic Server MIB tree, refer to the WebLogic Server SNMP MIB Reference. For more information about the structure of the MIB and its object identifiers (OIDs), refer to "Object Identifiers" in *WebLogic SNMP Management Guide*.

Syntax

```
java snmpwalk [-d] [-v (v1,v2)] [-c snmpCommunity] [-p snmpPort]
            [-t timeout] [-r retries] host OID
```

Argument	Definition
OID	Specifies the object ID of the node from which you want to retrieve a recursive list of object values. Start the value with '.'; otherwise, references are assumed to be relative to the standard MIB (`.1.3.6.1.2.1`), not the WebLogic Server MIB.

For information about the command arguments that are not listed in the above table, refer to Table 6-1, "Common Command Line Arguments," on page 6-3.

Example

The following example retrieves the name of all applications that have been deployed on the Administration Server. The OID in the example command is for the `applicationRuntimeName` object type, which represents the `Name` attribute of the `applicationRuntime` MBean.

```
java snmpwalk localhost .1.3.6.1.4.1.140.625.105.1.15
```

If you invoke this command from a computer that is running the example MedRecServer, the command returns output similar to the following truncated output. Note that the output includes the full OID for each instance of the `applicationRuntimeName` object type.

```
Object ID:
.1.3.6.1.4.1.140.625.105.1.15.32.49.102.98.97.100.97.102.99.57.48.50.
102.48.98.53.54.100.100.49.54.50.54.99.54.99.49.97.97.98.53.100.97
STRING: MedRecServer_uddiexplorer
```

```
Object ID:
.1.3.6.1.4.1.140.625.105.1.15.32.54.98.49.101.57.56.54.98.98.50.57.10
0.54.55.48.100.56.98.101.101.97.55.48.53.57.99.49.51.56.98.97.99
STRING: MedRecServer_StartupEAR

Object ID:
.1.3.6.1.4.1.140.625.105.1.15.32.56.48.97.53.50.52.99.101.53.54.57.54
.52.52.99.54.48.55.54.100.102.49.54.97.98.52.48.53.98.100.100.49
STRING: MedRecServer_wl_management_internal2

...
```

The following example retrieves the name of all applications that have been deployed on all servers in the medrec domain. The OID specified in the example command is the numerical value that the WebLogic Server MIB assigns to the applicationRuntimeName object type.

```
java snmpwalk -c public@medrec localhost .1.3.6.1.4.1.140.625.105.1.15
```

The following example returns all attributes of the ServerRuntimeMBean instance that is hosted on a Managed Server named MS1. Note that the OID .1.3.6.1.4.1.140.625.360 refers to the serverRuntimeTable object in the WebLogic MIB.

```
java snmpwalk -c public@MS1 localhost .1.3.6.1.4.1.140.625.360
```

snmpgetnext

Returns a description of the managed object that immediately follows one or more OIDs that you specify.

Instead of the recursive listing that the snmpwalk command provides, this command returns the description of only the one managed object whose OID is the next in sequence. You could string together a series of snmpgetnext commands to achieve the same result as the snmpwalk command.

If you specify an object type, this command returns the first instance of the object type, regardless of how many instances of the type exist.

To see the WebLogic Server MIB tree, refer to the WebLogic Server SNMP MIB Reference. For information about the structure of the MIB and its object identifiers (OIDs), refer to “Object Identifiers” in *WebLogic SNMP Management Guide*.

Syntax

```
java snmpgetnext [-d] [-v (v1,v2)] [-c snmpCommunity] [-p snmpPort]
             [-t timeout] [-r retries] host OID [OID]...
```

Argument	Definition
OID [*OID*] ...	Specifies one or more object IDs. You can specify an OID for an object type or an object instance. Start the values with '.'; otherwise, references are assumed to be relative to the standard MIB (.1.3.6.1.2.1), not the WebLogic Server MIB.

For information about the command arguments that are not listed in the above table, refer to Table 6-1, "Common Command Line Arguments," on page 6-3.

Example

The following example retrieves the name of an application that has been deployed on the Administration Server. The OID in the example command is for the `applicationRuntimeName` object type, which represents the `Name` attribute of the `applicationRuntime` MBean.

```
java snmpgetnext localhost .1.3.6.1.4.1.140.625.105.1.15
```

If you invoke this command from a computer that is running the example MedRecServer, the command returns output similar to the following:

```
Response PDU received from /127.0.0.1, community: public
Object ID:
.1.3.6.1.4.1.140.625.105.1.15.32.49.102.98.97.100.97.102.99.57.48.50.
102.48.98.53.54.100.100.49.54.50.54.99.54.99.49.97.97.98.53.100.97
STRING: MedRecServer_uddiexplorer
```

To determine whether there are additional applications deployed on the Administration Server, you can use the output of the initial `snmpgetnext` command as input for an additional `snmpgetnext` command:

```
java snmpgetnext localhost
.1.3.6.1.4.1.140.625.105.1.15.32.49.102.98.97.100.97.102.99.57.48.50.102.
48.98.53.54.100.100.49.54.50.54.99.54.99.49.97.97.98.53.100.97
```

The command returns output similar to the following:

```
Response PDU received from /127.0.0.1, community: public
Object ID:
.1.3.6.1.4.1.140.625.105.1.15.32.54.98.49.101.57.56.54.98.98.50.57.10
```

```
0.54.55.48.100.56.98.101.101.97.55.48.53.57.99.49.51.56.98.97.99
STRING: MedRecServer_StartupEAR
```

The following example specifies two OIDs to retrieve the name of an application that has been deployed on the Administration Server **and** the name of a JDBC connection pool. The OIDs in the example command are for the `applicationRuntimeName` object type, which represents the `Name` attribute of the `ApplicationRuntime` MBean, and `jdbcConnectionPoolRuntimeName`, which represents the `Name` attribute of the `JDBCConnectionPoolRuntimeMBean`.

```
java snmpgetnext localhost .1.3.6.1.4.1.140.625.105.1.15
.1.3.6.1.4.1.140.625.190.1.15
```

If you invoke this command from a computer that is running the example MedRecServer, the command returns output similar to the following:

```
Response PDU received from /127.0.0.1, community: public
Object ID:
.1.3.6.1.4.1.140.625.105.1.15.32.49.102.98.97.100.97.102.99.57.48.50.
102.48.98.53.54.100.100.49.54.50.54.99.54.99.49.97.97.98.53.100.97
STRING: MedRecServer_uddiexplorer
Object ID:
.1.3.6.1.4.1.140.625.190.1.15.32.53.53.49.48.50.55.52.57.57.49.99.102
.55.48.98.53.50.54.100.48.100.53.53.52.56.49.57.49.49.99.99.99
STRING: MedRecPool-PointBase
```

snmpget

Retrieves the value of one or more object instances. This command does not accept OIDs for object types.

Syntax

```
java snmpget [-d] [-v (v1,v2)] [-c snmpCommunity] [-p snmpPort]
             [-t timeout] [-r retries] host object-instance-OID
             [object-instance-OID]...
```

Argument	Definition
object-instance-OID [*object-instance-OID*]...	The object ID of an object instance. This command does not accept OIDs for object types. Start the value with '.'; otherwise, references are assumed to be relative to the standard MIB, not the WebLogic Server MIB.

Example

The following example retrieves the value of the `serverRuntimeState` and `serverRuntimeListenPort` attribute instances for the Administration Server.

```
java snmpget localhost
.1.3.6.1.4.1.140.625.360.1.60.32.102.100.48.98.101.102.100.99.102.52.98.
97.48.48.49.102.57.53.51.50.100.102.53.55.97.101.52.56.99.99.97.99
.1.3.6.1.4.1.140.625.360.1.35.32.102.100.48.98.101.102.100.99.102.52.
98.97.48.48.49.102.57.53.51.50.100.102.53.55.97.101.52.56.99.99.97.99
```

If you invoke this command from a computer that is running the example MedRecServer, the command returns output similar to the following:

```
Response PDU received from /127.0.0.1, community: public
Object ID:
.1.3.6.1.4.1.140.625.360.1.60.32.102.100.48.98.101.102.100.99.102.52.
98.97.48.48.49.102.57.53.51.50.100.102.53.55.97.101.52.56.99.99.97.99
STRING: RUNNING
Object ID:
.1.3.6.1.4.1.140.625.360.1.35.32.102.100.48.98.101.102.100.99.102.52.
98.97.48.48.49.102.57.53.51.50.100.102.53.55.97.101.52.56.99.99.97.99
INTEGER: 7001
```

Commands for Testing Traps

Table 6-3 is an overview of commands that generate and receive traps for testing purposes.

Table 6-3 Overview of Commands for Retrieving Information about WebLogic Server

Command	Description
`snmpv1trap`	Constructs an SNMPv1 trap and distributes it to the SNMP manager or trap daemon that is running on the specified host and listening on the specified port number. See "snmpv1trap" on page 6-10.
`snmptrapd`	Starts a daemon that receives traps and prints information about the trap. See "snmptrapd" on page 6-13.

snmpv1trap

Constructs an SNMPv1 trap and distributes it to the SNMP manager or trap daemon that is running on the specified host and listening on the specified port number. For more information about the trap daemon, refer to "snmptrapd" on page 6-13.

As part of invoking this command, you specify the value for fields within the trap packet that you want to send. The values that you specify must resolve to traps that are defined in the WebLogic Server MIB. For information about WebLogic Server traps and the fields that trap packets require, refer to "Format of WebLogic Trap Notifications" in the *WebLogic SNMP Management Guide*.

Syntax

```
java snmpv1trap [-d] [-c snmpCommunity] [-p TrapDestinationPort]
          TrapDestinationHost .1.3.6.1.4.140.625
          agent-addr generic-trap specific-trap timestamp
          [OID {INTEGER | STRING | GAUGE | TIMETICKS | OPAQUE |
          IPADDRESS | COUNTER} value] ...
```

Argument	Definition
`-c` *`snmpCommunity`*	Specifies a password (community name) that secures the data in the trap. If you do not specify a value, the command assumes `-c public`.
`-p` *`TrapDestinationPort`*	Specifies the port number on which the SNMP manager or trap daemon is listening. If you do not specify a value, the command assumes `-p 162`.
`TrapDestinationHost`	Specifies the DNS name or IP address of the computer that hosts the SNMP manager or trap daemon.
`.1.3.6.1.4.140.625`	Specifies the value of the trap's `enterprise` field, which contains the beginning portion of the OID for all WebLogic Server traps.
`agent-addr`	Specifies the value of the trap's `agent address` field. This field is intended to indicate the computer on which the trap was generated. When using the `snmpv1trap` command to generate a trap, you can specify any valid DNS name or IP address.
`generic-trap`	Specifies the value of the trap's `generic trap type` field. For a list of valid values, refer to "Format of WebLogic Trap Notifications" in the *WebLogic SNMP Management Guide*.
`specific-trap`	Specifies the value of the trap's `specific trap type` field. For a list of valid values, refer to "Format of WebLogic Trap Notifications" in the *WebLogic SNMP Management Guide.*

Argument	Definition
timestamp	Specifies the value of the trap's timestamp field. This field is intended to indicate the length of time between the last re-initialization of the SNMP agent and the time at which the trap was issued. When using the snmpv1trap command to generate a trap, any number of seconds is sufficient.
OID {INTEGER | STRING | GAUGE | TIMETICKS | OPAQUE | IPADDRESS | COUNTER} *value*	(Optional) Specifies the value of the trap's variable bindings field, which consists of name/value pairs that further describe the trap notification. For each name/value pair, specify an OID, a value type, and a value. For example, a log message trap includes a trapTime binding to indicate the time at which the trap is generated. To include this variable binding in the test trap that you generate, specify the OID for the trapTime variable binding, the STRING keyword, and a string that represents the time: .1.3.6.1.4.1.140.625.100.5 STRING "2:00 pm"

Example

The following example generates a log message trap that contains the trapTime and trapServerName variable bindings. It broadcasts the trap through port 165. In the example:

- 6 is the generic trap value that specifies "other WebLogic Server traps."
- 60 is the specific trap value that WebLogic Server uses to identify log message traps.
- .1.3.6.1.4.1.140.625.100.5 is the OID for the trapTime variable binding and .1.3.6.1.4.1.140.625.100.10 is the OID for the trapServerName variable binding.

```
java snmpv1trap -p 165 localhost .1.3.6.1.4.140.625 localhost 6 60 1000
.1.3.6.1.4.1.140.625.100.5 STRING "2:00 pm" .1.3.6.1.4.1.140.625.100.10
STRING localhost
```

The SNMP manager (or trap daemon) that is listening at port number 165 receives the trap. If the trap daemon is listening on 165, it returns the following:

```
Trap received from: /127.0.0.1, community: public
Enterprise: .1.3.6.1.4.140.625
```

```
Agent: /127.0.0.1
TRAP_TYPE: 6
SPECIFIC NUMBER: 60
Time: 1000
VARBINDS:
Object ID: .1.3.6.1.4.1.140.625.100.5
STRING: 2:00 pm
Object ID: .1.3.6.1.4.1.140.625.100.10
STRING: localhost
```

snmptrapd

Starts a daemon that receives traps and prints information about the trap.

Syntax

```
java snmptrapd [-d] [-c snmpCommunity] [-p TrapDestinationPort]
```

Argument	Definition
`-c` *`snmpCommunity`*	Specifies that community name that the SNMP agent (or `snmpv1trap` command) used to generate the trap. If you do not specify a value, the command assumes `-c public`.
`-p` *`TrapDestinationPort`*	Specifies the port number on which the trap daemon receives traps. If you do not specify a value, the command assumes `-p 162`.

Example

The following command starts a trap daemon and instructs it to listen for requests on port 165. The daemon runs in the shell until you kill the process or exit the shell:

```
java snmptrapd -p 165
```

If the command succeeds, the trap daemon returns a blank line with a cursor. The trap daemon waits in this state until it receives a trap, at which point it prints the trap.

Example: Sending Traps to the Trap Daemon

To generate WebLogic Server traps and receive them through the trap daemon:

1. Open a command prompt (shell) and invoke the following script:

   `WL_HOME\server\bin\setWLSEnv.sh` (or `setWLSEnv.cmd` on Windows)
   where `WL_HOME` is the directory in which you installed WebLogic Server.

2. To start the trap daemon, enter the following command:

   ```
   java snmptrapd
   ```

3. Open another shell and invoke the following script:

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

4. To generate a trap, enter the following command:

   ```
   java snmpv1trap localhost .1.3.6.1.4.140.625 localhost 6 60 1000
   ```

The `snmpv1trap` command generates a `serverStart` Trap and broadcasts it through port 162.

In the shell in which the trap daemon is running, the daemon prints the following:

```
Trap received from: /127.0.0.1, community: public
Enterprise: .1.3.6.1.4.140.625
Agent: /127.0.0.1
TRAP_TYPE: 6
SPECIFIC NUMBER: 60
Time: 1000
VARBINDS:
```

Index

A

Administration commands, overview 1-20, 1-39, 1-52, 6-4, 6-10
Administration Console
    specifying private key password for use with SSL 4-16
Administration Server
    starting from command line 4-1
appc 3-30

C

CANCEL_SHUTDOWN, WebLogic Server command 1-21
Command-line interface
    administration commands overview 1-20, 1-39, 1-52, 6-4, 6-10
    command syntax and arguments 1-2, 6-2
    Mbean management commands overview 1-63, 1-81
CONNECT, WebLogic Server command 1-21, 1-24
Connection Pool Administration commands, overview 1-52
CREATE, WebLogic Server command 1-64
CREATE_POOL, WebLogic Server command 1-53
Creating Mbeans, CREATE command 1-64
customer support contact information xvi

D

DDInit 3-12
DELETE, WebLogic Server command 1-66
Deleting Mbeans, DELETE command 1-66
DESTROY_POOL, WebLogic Server command 1-56
DISABLE_POOL, WebLogic Server command 1-57
documentation, where to find it xvi

E

ENABLE_POOL, WebLogic Server command 1-58
EXISTS_POOL, WebLogic Server command 1-61

G

GET, WebLogic Server command 1-68
Getting help for a WebLogic Server command 1-43
Getting Mbean information, GET command 1-68

H

HELP, WebLogic Server command 1-43
host name verification
    default configuration 1-6
    using 1-6
Host Name Verifier
    disabling at start-up 4-15
    specifying a custom 4-15

I

INVOKE, WebLogic Server command 1-70

J

Java heap memory
- specifying minimum and maximum 4-6

JHTML 3-20
jhtml2jsp 3-20
JNDI naming tree
- list node bindings 1-44

L

LICENSES, WebLogic Server command 1-44
LIST, WebLogic Server command 1-44
Listening ports, verify 1-29
LOCK, WebLogic Server command 1-28

M

Mbean management commands, overview 1-63, 1-81

P

PING, WebLogic Server command 1-29
printing product documentation xvi

R

RESET_POOL, WebLogic Server command 1-60
Resetting connection pools, RESET_POOL command 1-60

S

Secure Sockets Layer (SSL)
- host name verification 1-6

server name
- specifying at startup 4-24

SERVERLOG, WebLogic Server command 1-47
SET, WebLogic Server command 1-75
Setting attribute values, SET command 1-75
SHUTDOWN, WebLogic Server command 1-30
SSL
- specifying private key password at server startup 4-16

SSL session caching
- indicating 4-16

support
- technical xvi

system home directory, WebLogic
- specifying at startup 4-7

T

THREAD_DUMP, WebLogic Server command 1-49
Threads, view running 1-49

U

UNLOCK, WebLogic Server command 1-39

V

Verify WebLogic Server listening ports 1-29
VERSION, WebLogic Server command 1-50
Viewing server log files, SERVERLOG command 1-47

W

WebLogic Server
- licenses, viewing 1-44
- specifying user name of at startup 4-18

WebLogic Server commands
- administration commands overview 1-20, 1-39, 1-52, 6-4, 6-10
- CANCEL_SHUTDOWN 1-21
- CONNECT 1-21, 1-24
- connection pool commands overview 1-52

CREATE 1-64
CREATE_POOL 1-53
DELETE 1-66
DESTROY_POOL 1-56
DISABLE_POOL 1-57
ENABLE_POOL 1-58
EXISTS_POOL 1-61
GET 1-68
HELP 1-43
INVOKE 1-70
LICENSES 1-44
LIST 1-44
LOCK 1-28
Mbean management commands overview 1-63, 1-81
PING 1-29
RESET_POOL 1-60
SERVERLOG 1-47
SET 1-75
SHUTDOWN 1-30
syntax and arguments 1-2, 6-2
THREAD_DUMP 1-49
UNLOCK 1-39
VERSION 1-50

WebLogicObjectName
defined 1-72

wlstart 5-2