## Contents

1. Introduction and Roadmap
   - Document Scope and Audience ........................................ 1-1
   - Guide to This Document ............................................ 1-1
   - Related Documentation ............................................... 1-2
   - New and Changed Features in This Release ....................... 1-2

2. Using the Oracle WebLogic Server Java Utilities
   - appc ................................................................. 2-3
   - AppletArchiver ....................................................... 2-3
   - Syntax ........................................................................ 2-3
   - autotype (deprecated) .................................................. 2-4
   - BuildXMLGen ............................................................ 2-4
   - CertGen ................................................................. 2-4
   - Syntax ........................................................................ 2-4
   - Example ...................................................................... 2-6
   - ClientDeployer ........................................................... 2-7
   - clientgen ................................................................. 2-7
   - Conversion (deprecated) ................................................ 2-7
   - dbping ................................................................. 2-8
   - Creating a DB2 Package with dbping ............................... 2-8
   - Syntax ...................................................................... 2-8
   - Example .................................................................... 2-10
<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ddcreate (deprecated)</td>
<td>2-11</td>
</tr>
<tr>
<td>DDInit</td>
<td>2-11</td>
</tr>
<tr>
<td>Deployer</td>
<td>2-12</td>
</tr>
<tr>
<td>der2pem</td>
<td>2-12</td>
</tr>
<tr>
<td>Syntax</td>
<td>2-12</td>
</tr>
<tr>
<td>Example</td>
<td>2-13</td>
</tr>
<tr>
<td>ejbc (deprecated)</td>
<td>2-13</td>
</tr>
<tr>
<td>EJBGem</td>
<td>2-14</td>
</tr>
<tr>
<td>encrypt</td>
<td>2-14</td>
</tr>
<tr>
<td>Syntax</td>
<td>2-14</td>
</tr>
<tr>
<td>Examples</td>
<td>2-15</td>
</tr>
<tr>
<td>getProperty</td>
<td>2-15</td>
</tr>
<tr>
<td>Syntax</td>
<td>2-16</td>
</tr>
<tr>
<td>Example</td>
<td>2-16</td>
</tr>
<tr>
<td>host2ior</td>
<td>2-16</td>
</tr>
<tr>
<td>Syntax</td>
<td>2-16</td>
</tr>
<tr>
<td>ImportPrivateKey</td>
<td>2-17</td>
</tr>
<tr>
<td>Syntax</td>
<td>2-17</td>
</tr>
<tr>
<td>Example</td>
<td>2-18</td>
</tr>
<tr>
<td>jhtml2jsp</td>
<td>2-19</td>
</tr>
<tr>
<td>Syntax</td>
<td>2-19</td>
</tr>
<tr>
<td>jspc (deprecated)</td>
<td>2-20</td>
</tr>
<tr>
<td>logToZip</td>
<td>2-20</td>
</tr>
<tr>
<td>Syntax</td>
<td>2-20</td>
</tr>
<tr>
<td>Examples</td>
<td>2-21</td>
</tr>
<tr>
<td>MBean Commands</td>
<td>2-21</td>
</tr>
<tr>
<td>MulticastTest</td>
<td>2-21</td>
</tr>
<tr>
<td>Syntax</td>
<td>2-22</td>
</tr>
<tr>
<td>Command</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------</td>
<td>------</td>
</tr>
<tr>
<td>myip</td>
<td>2-23</td>
</tr>
<tr>
<td>pem2der</td>
<td>2-23</td>
</tr>
<tr>
<td>pointbase</td>
<td>2-24</td>
</tr>
<tr>
<td>rmie</td>
<td>2-24</td>
</tr>
<tr>
<td>servicegen (deprecated)</td>
<td>2-25</td>
</tr>
<tr>
<td>SearchAndBuild</td>
<td>2-26</td>
</tr>
<tr>
<td>source2wsdd (deprecated)</td>
<td>2-26</td>
</tr>
<tr>
<td>system</td>
<td>2-27</td>
</tr>
<tr>
<td>ValidateCertChain</td>
<td>2-28</td>
</tr>
<tr>
<td>verboseToZip</td>
<td>2-28</td>
</tr>
<tr>
<td>wlappe</td>
<td>2-29</td>
</tr>
<tr>
<td>wlmdeploy</td>
<td>2-30</td>
</tr>
<tr>
<td>wlpackage</td>
<td>2-30</td>
</tr>
</tbody>
</table>
3. weblogic.Server Command-Line Reference

Required Environment and Syntax for weblogic.Server ................................. 3-2
Environment ....................................................................................... 3-2
Modifying the Classpath ................................................................. 3-2
Syntax .............................................................................................. 3-3
Default Behavior .............................................................................. 3-3
weblogic.Server Configuration Options ............................................... 3-5
JVM Parameters ............................................................................... 3-5
Location of Configuration Data .......................................................... 3-6
Example ........................................................................................... 3-8
Options that Override a Server’s Configuration ...................................... 3-9
Server Communication ...................................................................... 3-10
SSL .................................................................................................... 3-13
Security .............................................................................................. 3-17
Message Output and Logging ............................................................. 3-21
Other Server Configuration Options .................................................. 3-23
Clusters .............................................................................................. 3-26
Deployment .......................................................................................... 3-26
Using the weblogic.Server Command Line to Start a Server Instance ........ 3-26
Using the weblogic.Server Command Line to Create a Domain .............. 3-27
Verifying Attribute Values That Are Set on the Command Line ............... 3-29
4. WebLogic SNMP Agent Command-Line Reference (Deprecated)

Required Environment for the SNMP Command-Line Interface ............... 4-2
Syntax and Common Arguments for the SNMP Command-Line Interface .......... 4-2
Commands for Retrieving WebLogic Server Managed Objects ................... 4-4
snmpwalk .................................................. 4-4
Syntax ..................................................... 4-5
Example .................................................... 4-5
snmpgetnext ................................. 4-6
Syntax ..................................................... 4-6
Example .................................................... 4-6
snmpget ................................................... 4-8
Syntax ..................................................... 4-8
Example .................................................... 4-8
Commands for Testing Traps ........................................... 4-9
snmptrapd .................................................. 4-9
Syntax ..................................................... 4-9
Example .................................................... 4-10
snmpv1trap ............................................... 4-10
Syntax ..................................................... 4-10
Example .................................................... 4-12
Example: Using snmpv1trap to Send Traps to the Trap Daemon ............... 4-13
Example: Using the WebLogic SNMP Agent to Send Traps to the Trap Daemon . 4-14
CHAPTER 1

Introduction and Roadmap

This section describes the contents and organization of this guide—Oracle WebLogic Server Command Reference.

- “Document Scope and Audience” on page 1-1
- “Guide to This Document” on page 1-1
- “Related Documentation” on page 1-2
- “New and Changed Features in This Release” on page 1-2

Document Scope and Audience

This document describes Oracle WebLogic Server command-line reference features and Java utilities and how to use them to administer Oracle WebLogic Server.

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

Guide to This Document

The document is organized as follows:

- This chapter, “Introduction and Roadmap,” describes the scope of this guide and lists related documentation.
Introduction and Roadmap

- Chapter 2, “Using the Oracle WebLogic Server Java Utilities,” describes various Java utilities you can use to manage and troubleshoot an Oracle WebLogic Server domain.


- Chapter 4, “WebLogic SNMP Agent Command-Line Reference (Deprecated),” describes using Simple Network Management Protocol (SNMP) to communicate with enterprise-wide management systems.

Related Documentation

- Using Ant Tasks to Configure and Use a WebLogic Server Domain in Developing Applications with Oracle WebLogic Server.

- WebLogic Scripting Tool

- Configuring WebLogic Server Environments

- Administration Console Online Help

New and Changed Features in This Release

For a comprehensive listing of the new WebLogic Server features introduced in this release, see “What’s New in WebLogic Server” in the Release Notes.
CHAPTER 2

Using the Oracle WebLogic Server Java Utilities

Oracle 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 3-2.

Oracle WebLogic Server provides several Java programs that simplify installation and configuration tasks, provide services, and offer convenient shortcuts. The Java utilities provided with Oracle WebLogic Server are all described below. The command-line syntax is specified for all utilities and, for some, examples are provided.

Oracle 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 2-3
- “AppletArchiver” on page 2-3
- “autotype (deprecated)” on page 2-4
- “BuildXMLGen” on page 2-4
- “CertGen” on page 2-4
- “ClientDeployer” on page 2-7
Using the Oracle WebLogic Server Java Utilities

- “clientgen” on page 2-7
- “Conversion (deprecated)” on page 2-7
- “dbping” on page 2-8
- “DDInit” on page 2-11
- “Deployer” on page 2-12
- “der2pem” on page 2-12
- “ejbc (deprecated)” on page 2-13
- “EJBGen” on page 2-14
- “encrypt” on page 2-14
- “getProperty” on page 2-15
- “host2ior” on page 2-16
- “ImportPrivateKey” on page 2-17
- “jscp (deprecated)” on page 2-20
- “logToZip” on page 2-20
- “MBean Commands” on page 2-21
- “MulticastTest” on page 2-21
- “myip” on page 2-23
- “pem2der” on page 2-23
- “rmic” on page 2-24
- “Schema” on page 2-24
- “source2wsdd (deprecated)” on page 2-26
- “system” on page 2-27
- “ValidateCertChain” on page 2-28
- “verboseToZip” on page 2-28
- “wlappc” on page 2-29
The appc compiler generates and compiles the classes needed to deploy EJBs and JSPs to Oracle 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 Reference in Programming WebLogic 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
```

#### Table 2-1 AppletArchiver Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>URL</td>
<td>URL for the applet.</td>
</tr>
<tr>
<td>filename</td>
<td>Local filename that is the destination for the .jar/.cab archive.</td>
</tr>
</tbody>
</table>
autotype (deprecated)

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.

Web Services are now a Java EE standard, which has resulted in many changes between 8.1 and 9.0 WebLogic Web Services. For information about what has changed between 8.1 and 9.0 Web Services, and what has been deprecated, see Differences Between 8.1 and 9.0 WebLogic Web Services in Programming Web Services for WebLogic Server.

For a complete list of Web Services Ant tasks, see Ant Task Reference in WebLogic Web Services: Reference.

BuildXMLGen

Use BuildXMLGen to generate a build.xml file for enterprise applications in the split-directory structure. For complete documentation of this utility, see Building Applications in a Split Development Directory in Developing Applications with WebLogic Server.

CertGen

The CertGen utility generates certificates that should only be used for demonstration or testing purposes, not in a production environment.

Syntax

$ java utils.CertGen

- -certfile <cert_file> -keyfile <private_key_file>
- -keyfilepass <private_key_password>
[ -cacert <ca_cert_file>][ -cakey <ca_key_file>]
[ -cakeypass <ca_key_password>]
[ -selfsigned] [ -strength <key_strength>]
[ -e <email_address>][ -cn <common_name>]
[ -ou <org_unit>][ -o <organization>]
[ -l <locality>][ -s <state>][ -c <country_code>]
[ -keyusage [digitalSignature,nonRepudiation,keyEncipherment,
dataEncipherment,keyAgreement,keyCertSign,
cRLSign,encipherOnly,decipherOnly]]
[-keyusagecritical true|false]
[-subjectkeyid <subject_key_identifier>]
[-subjectkeyidformat UTF-8|BASE64]
[-help]

### Table 2-2 CertGen Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-certfile</code> cert_file</td>
<td>Respectively, the output file names without extensions for the generated public certificate and private key. The appropriate extensions are appended when the pem and der files are created.</td>
</tr>
<tr>
<td><code>-keyfile</code> private_key_file</td>
<td></td>
</tr>
<tr>
<td><code>-keyfilepass</code> private_key_password</td>
<td>The password for the generated private key.</td>
</tr>
<tr>
<td><code>-cacert</code> ca_cert_file</td>
<td>Respectively, the public certificate, private key file, and private key password of the CA that will be used as the issuer of the generated certificate.</td>
</tr>
<tr>
<td><code>-cakey</code> ca_key_file</td>
<td>If one or more of these options are not specified, the relevant demonstration CA files will be used: CertGenCA.der and CertGenCAKey.der.</td>
</tr>
<tr>
<td><code>-cakeypass</code> ca_key_password</td>
<td>The CertGen utility first looks in the current working directory, then in the WL_HOME/lib directory.</td>
</tr>
<tr>
<td><code>-selfsigned</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>-strength</code> key_strength</td>
<td>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.</td>
</tr>
<tr>
<td><code>-e</code> email_address</td>
<td>The email address associated with the generated certificate.</td>
</tr>
<tr>
<td><code>-cn</code> common_name</td>
<td>The name associated with the generated certificate.</td>
</tr>
<tr>
<td><code>-ou</code> org_unit</td>
<td>The name of the organizational unit associated with the generated certificate.</td>
</tr>
</tbody>
</table>
Table 2-2 CertGen Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>-o organization</td>
<td>The name of the organization associated with the generated certificate.</td>
</tr>
<tr>
<td>-l locality</td>
<td>The name of a city or town.</td>
</tr>
<tr>
<td>-s state</td>
<td>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.</td>
</tr>
<tr>
<td>-c country_code</td>
<td>Two-letter ISO code for your country. The code for the United States is US.</td>
</tr>
<tr>
<td>-keyusage</td>
<td>Generate certificate with a keyusage extension, and with bits set according to the comma-separated list of bit names. Specify a key usage when you want to restrict the operation for a key that could be used for more than one operation.</td>
</tr>
<tr>
<td>-keyusagecritical true</td>
<td>false</td>
</tr>
<tr>
<td>-subjectkeyid</td>
<td>Generates a certificate with the specified subject key identifier.</td>
</tr>
<tr>
<td>-subjectkeyidformat UTF-8</td>
<td>BASE64</td>
</tr>
</tbody>
</table>

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 -keyfilepass mykeypass -certfile testcert -keyfile testkey
```
Generating a certificate with common name return and key strength 1024 issued by CA with certificate from CertGenCA.der file and key from CertGenCAKey.der file

ClientDeployer

You use weblogic.ClientDeployer to extract the client-side JAR file from a Java EE EAR file, creating a deployable JAR file. The weblogic.ClientDeployer class is executed on the Java command line with the following syntax:

```java
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
java weblogic.ClientDeployer app.ear myclient
```

where `app.ear` is the EAR file that contains a Java EE 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
java weblogic.j2eeclient.Main clientjar URL [application args]
```

For example:

```java
java weblogic.j2eeclient.Main helloWorld.jar t3://localhost:7001
Greetings
```

clientgen

Use clientgen to generate the client-side artifacts, such as the JAX-RPC stubs, needed to invoke a Web Service. See Ant Task Reference in WebLogic Web Services: Reference.

Conversion (deprecated)

WebLogic Server 9.0 does not support conversion or upgrading from a pre-6.0 version of Oracle WebLogic Server. To upgrade from version 6.1 or later, see Upgrading WebLogic Application Environments.
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 Using Third-Party Drivers with WebLogic Server 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
```
### Table 2-3 dbping Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DBMS</strong></td>
<td>Varies by DBMS and JDBC driver:</td>
</tr>
<tr>
<td>DB2B</td>
<td>WebLogic Type 4 JDBC Driver for DB2</td>
</tr>
<tr>
<td>JCONNECT2</td>
<td>Sybase JConnect 5.5 (JDBC 2.0) driver</td>
</tr>
<tr>
<td>JCONNECT3</td>
<td>Sybase JConnect 6.0 (JDBC 2.0) driver</td>
</tr>
<tr>
<td>JCONNECT</td>
<td>Sybase JConnect driver</td>
</tr>
<tr>
<td>INFORMIXB</td>
<td>WebLogic Type 4 JDBC Driver for Informix</td>
</tr>
<tr>
<td>MSSQLSERVER4</td>
<td>WebLogic jDriver for Microsoft SQL Server</td>
</tr>
<tr>
<td>MSSQLSERVERB</td>
<td>WebLogic Type 4 JDBC Driver for Microsoft SQL Server</td>
</tr>
<tr>
<td>MYSQL</td>
<td>MySQL’s Type 4 Driver</td>
</tr>
<tr>
<td>ORACLE</td>
<td>WebLogic jDriver for Oracle</td>
</tr>
<tr>
<td>ORACLEB</td>
<td>WebLogic Type 4 JDBC Driver for Oracle</td>
</tr>
<tr>
<td>ORACLE_THIN</td>
<td>Oracle Thin Driver</td>
</tr>
<tr>
<td>POINTBASE</td>
<td>PointBase Universal Driver</td>
</tr>
<tr>
<td>SYBASEB</td>
<td>WebLogic Type 4 JDBC Driver for Sybase</td>
</tr>
<tr>
<td><strong>[-d dynamicSections]</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>user</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>
Using the Oracle WebLogic Server Java Utilities

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 (deprecated)**

This Ant task calls EARInit, which generates an application.xml and a weblogic-application.xml file for an EAR. For more information, see “EarInit (deprecated)” on page 2-12.

**DDInit**

DDInit is a utility for generating deployment descriptors for applications to be deployed on Oracle 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.

In its command-line version, DDInit writes new files that overwrite existing descriptor files. If META-INF or WEB-INF does not exist, DDInit creates it.

Specify the type of Java EE deployable unit (either Web Application or Enterprise Application) for which you want deployment descriptors generated by using the DDInit command specific to the type, as described below.

**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.

    prompt> java weblogic.marathon.ddinit.WebInit <module>
EarInit (deprecated)
The EarInit tool is deprecated in this version of Oracle WebLogic Server. As a result, you should not:

- Use the DDInit utility to generate deployment descriptors for Enterprise applications.
- Use the ddcreate ant task, which calls 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.

```
prompt> java weblogic.marathon.ddinit.EarInit <module>
```

Deployer

Using the weblogic.Deployer tool, you can deploy Java EE applications and components to WebLogic Servers in a command-line or scripting environment. For detailed information on using this tool, see weblogic.Deployer Command-Line Reference in Deploying Applications to Oracle WebLogic Server.

The weblogic.Deployer utility replaces the weblogic.deploy utility, which has been deprecated.

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]
```
Table 2-4 der2pem Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>derFile</td>
<td>The name of the file to convert. The filename must end with a .der extension, and must contain a valid certificate in .der format.</td>
</tr>
<tr>
<td>headerFile</td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>• “-----BEGIN RSA PRIVATE KEY-----” for an unencrypted private key.</td>
</tr>
<tr>
<td></td>
<td>• “-----BEGIN ENCRYPTED PRIVATE KEY-----” for an encrypted private key.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> There must be a new line at the end of the header line in the file.</td>
</tr>
<tr>
<td>footerFile</td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>• “-----END RSA PRIVATE KEY-----” for an unencrypted private key.</td>
</tr>
<tr>
<td></td>
<td>• “-----END ENCRYPTED PRIVATE KEY-----” for an encrypted private key.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> There must be a new line at the end of the header line in the file.</td>
</tr>
</tbody>
</table>

Example

$ java utils.der2pem graceland_org.der
Decoding

.................................................................

**ejbc (deprecated)**

See [apce Reference](#) in *Programming Weblogic 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 Enterprise JavaBeans.

encrypt

The weblogic.security.Encrypt utility encrypts cleartext strings for use with Oracle WebLogic Server. The utility uses the encryption service of the current directory, or the encryption service for a specified Oracle WebLogic Server domain root directory.

Note: An encrypted string must have been encrypted by the encryption service in the Oracle 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 an Oracle WebLogic Server domain; it cannot be run from a client.

Note: Oracle recommends running the utility from 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 ]
```
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}I2hsIIn56KKKs3
```

getProperty

The `getProperty` utility gives you details about your Java setup and your system. It takes no arguments.
Using the Oracle WebLogic Server Java Utilities

**Syntax**

$ java utils.getProperty

**Example**

$ java utils.getProperty
   -- listing properties --
   user.language=en
   java.home=c:\javall\bin\..
   awt.toolkit=sun.awt.windows.WToolkit
   file.encoding.pkg=sun.io
   java.version=1.1_Final
   file.separator=\n
   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 an Oracle 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**

```sh
$ java utils.ImportPrivateKey
-certfile <cert_file> -keyfile <private_key_file>
[-keyfilepass <private_key_password>]
-keystore <keystore> -storepass <storepass> [-storetype <storetype>]
-alias <alias> [-keypass <keypass>]
[-help]
```

**Table 2-6 ImportPrivateKey Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>cert_file</td>
<td>The name of the certificate associated with the private key.</td>
</tr>
<tr>
<td>private_key_file</td>
<td>The name of the generated private key file.</td>
</tr>
<tr>
<td>private_key_password</td>
<td>The password for the private key.</td>
</tr>
<tr>
<td>keystore</td>
<td>The name of the keystore. A new keystore is created if one does not exist.</td>
</tr>
<tr>
<td>storepass</td>
<td>The password to open the keystore.</td>
</tr>
<tr>
<td>storetype</td>
<td>The type (format) of the keystore.</td>
</tr>
</tbody>
</table>

The `storetype` argument, which is the same as that used by the `keytool` command, specifies the type of Java keystore. The default `storetype` is `jks`, defined by the `keystore.type` property in the `java.security` file:

```java
keystore.type=jks
```

You can specify another `storetype` (for example, `pcks12` or `nCipher.SWorld`) if a configured security provider supports that type.
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 -keyfilepass mykeypass -certfile testcert -keyfile testkey
   Generating a certificate with common name return and key strength 1024 issued by CA with certificate from CertGenCA.der file and key from CertGenCAKey.der file
   ```

2. Convert the certificate from DER format to PEM format.

   ```
   $ java utils.der2pem CertGenCA.der
   ```

3. Concatenate the certificate and the Certificate Authority (CA).
4. Create a new keystore named mykeystore and load the private key located in the testkey.pem file.

```bash
$ cat testcert.pem CertGenCA.pem >> newcerts.pem

$ java utils.ImportPrivateKey -keystore mykeystore -storepass mypasswd -keyfile mykey -keyfilepass mykeypass -certfile newcerts.pem -keyfile testkey.pem -alias passalias
```

No password was specified for the key entry
Key file password will be used
Imported private key testkey.pem and certificate newcerts.pem into a new keystore mykeystore of type jks under alias passalias

---

**jhtml2jsp**

Converts JHTML files to JSP files. Be sure to inspect the results carefully. Given the unpredictability of the JHTML code, *jhtml2jsp* will not necessarily produce flawless translations.

The 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 reference these methods to access the `servlet context` or `config` objects, you must 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 must edit your Java code in the generated JSP page to change the variable name to something other than a reserved word.

**Syntax**

```bash
$ java weblogic.utils.jhtml2jsp -d <directory> filename.jhtml
```

or

```bash
$ java weblogic.utils.jhtml2jsp filename.jhtml
```
jspc (deprecated)

JSP-specific compiler task. Use “appc” on page 2-3.

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

Table 2-8 logToZip Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>logfile</td>
<td>Required. Fully-qualified pathname of the log file.</td>
</tr>
<tr>
<td>codebase</td>
<td>Required. Code base for the applet, or &quot;&quot; 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).</td>
</tr>
<tr>
<td>zipfile</td>
<td>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.</td>
</tr>
</tbody>
</table>
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, and SET) to administer MBeans. See Editing Commands in WebLogic Scripting Tool.

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 and hardware of the WebLogic Server host machine. For more information about configuring a cluster, see Using WebLogic Server Clusters.

**Syntax**

```bash
$ java utils.MulticastTest -n name -a address [-p portnumber] [-t timeout] [-s send]
```

**Table 2-9 MulticastTest Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>-n name</td>
<td>Required. A name that identifies the sender of the sequenced messages. Use a different name for each test process you start.</td>
</tr>
<tr>
<td>-a address</td>
<td>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.)</td>
</tr>
<tr>
<td>-p portnumber</td>
<td>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.)</td>
</tr>
<tr>
<td>-t timeout</td>
<td>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.</td>
</tr>
<tr>
<td>-s send</td>
<td>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.</td>
</tr>
</tbody>
</table>

**Example**

```bash
$ 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**

```bash
$ java utils.myip
```

**Example**

```bash
$ 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**

```bash
$ java utils.pem2der pemFile
```
Using the Oracle WebLogic Server Java Utilities

### pem2der Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pemFile</td>
<td>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.</td>
</tr>
</tbody>
</table>

### Example

```bash
$ java utils.pem2der graceland_org.pem
Decoding
.................................................................
.................................................................
.................................................................
.................................................................
.................................................................
.................................................................
```

### pointbase

PointBase is bundled with WebLogic Server as a sample database. Its documentation is also included at WL_HOME\common\eval\pointbase\docs, where WL_HOME is the WebLogic Server installation directory, typically C:\bea\wlserver_10.3.

### 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 Using the 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

Table 2-11 Schema Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>driverURL</td>
<td>Required. URL for the JDBC driver.</td>
</tr>
<tr>
<td>driverClass</td>
<td>Required. Pathname of the JDBC driver class.</td>
</tr>
<tr>
<td>-u username</td>
<td>Optional. Valid username.</td>
</tr>
<tr>
<td>-p password</td>
<td>Optional. Valid password for the user.</td>
</tr>
<tr>
<td>-verbose</td>
<td>Optional. Prints SQL statements and database messages.</td>
</tr>
<tr>
<td>SQLfile</td>
<td>Required. Text file with SQL statements.</td>
</tr>
</tbody>
</table>

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 (deprecated)

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.
Web Services are now a Java EE standard, which has resulted in many changes between 8.1 and 9.0 WebLogic Web Services. For information about what has changed between 8.1 and 9.0 Web Services, and what has been deprecated, see Differences Between 8.1 and 9.0 WebLogic Web Services in Programming Web Services for Weblogic Server.

For a complete list of Web Services Ant tasks, see Ant Task Reference in WebLogic Web Services: Reference.

**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**

```xml
<project name="all_modules" default="all" basedir="."/>
<taskdef name="buildAll" classname="weblogic.ant.taskdefs.build.SearchAndBuildTask"/>
<target name="all">
  <buildAll>
    <fileset dir="${basedir}"
      include="**uild.xml"/>
    <exclude name="build.xml"/>
  </fileset>
</target>
</project>
```

**source2wsdd (deprecated)**

Generates a web-services.xml deployment descriptor file from the Java source file for a Java class-implemented WebLogic Web Service.
Web Services are now a Java EE standard, which has resulted in many changes between 8.1 and 9.0 WebLogic Web Services. For information about what has changed between 8.1 and 9.0 Web Services, and what has been deprecated, see Differences Between 8.1 and 9.0 WebLogic Web Services in Programming Web Services for Weblogic Server.

For a complete list of Web Services Ant tasks, see Ant Task Reference in WebLogic Web Services: Reference.

**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**

```bash
$ java utils.system
```

**Example**

```bash
$ java utils.system
  * * * * * * java.version * * * * * *
  1.5.0_03
  * * * * * * java.vendor * * * * * *

  * * * * * * java.class.path * * * * * *
  C:\src_15003jr\bea\wlserver_10.3\server\classes;
  C:\dev\src\build\JROCKIT-2.0_0\lib\tools.jar;
  ...
  * * * * * * os.name * * * * * *
  Windows 2000
  * * * * * * os.arch * * * * * *
  x86
  * * * * * * os.version * * * * * *
  5.0
```
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
java utils.ValidateCertChain -file pemcertificatefilename
java utils.ValidateCertChain -pem pemcertificatefilename
java utils.ValidateCertChain -pkcs12store pkcs12storefilename
java utils.ValidateCertChain -pkcs12file pkcs12filename password
java utils.ValidateCertChain -jks alias storefilename [storePass]
```

Example of valid certificate chain:

```java
java utils.ValidateCertChain -pem zippychain.pem
Cert[0]: CN=zippy,OU=FOR TESTINGONLY,O=MyOrganization,L=MyTown,ST=MyState,C=US
Cert[1]: CN=CertGenCAB,OU=FOR TESTINGONLY,O=MyOrganization,L=MyTown,ST=MyState,C=US
Certificate chain appears valid
```

Example of invalid certificate chain:

```java
java utils.ValidateCertChain -jks mykey mykeystore
Cert[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=US
Certificate 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

```bash
$ java utils.verboseToZip inputFile zipfileToCreate
```
Example

$ java -verbose myapplication > & classList.tmp
$ java utils.verboseToZip classList.tmp app2.zip

wlappc

This utility compiles and validates a Java EE EAR file, an EJB JAR file, or a WAR file for deployment.

For more information, see Building Modules and Applications Using wlappc in Developing Applications with WebLogic Server.

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 Building Applications in a Split Development Directory in Developing Applications with WebLogic Server.

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 Using Ant Tasks to Configure a WebLogic Server Domain in Developing Applications with WebLogic Server.

Table 2-12  verboseToZip Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputFile</td>
<td>Required. Temporary file that contains the output of the application running in verbose mode.</td>
</tr>
<tr>
<td>zipFileToCreate</td>
<td>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.</td>
</tr>
</tbody>
</table>
Using the Oracle WebLogic Server Java Utilities

**wldeploy**

The `wldeploy` Ant task enables you to perform Deployer functions using attributes specified in an Ant task. See Deploying and Packaging from a Split Development Directory in Developing Applications with WebLogic Server.

**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 Deploying and Packaging from a Split Development Directory in Developing Applications with WebLogic Server.

**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 Developing Applications with Weblogic Server.

**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 Developing WebLogic Web Services Starting From a WSDL File: Main Steps in Programming Web Services for Weblogic Server.

**wsdlgen (deprecated)**

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. Web Services are now a Java EE standard, which has resulted in many changes between 8.1 and 9.0 WebLogic Web Services. For information about what has changed between 8.1 and 9.0 Web Services, and what has been deprecated, see Differences Between 8.1 and 9.0 WebLogic Web Services in Programming Web Services for Weblogic Server.
For a complete list of Web Services Ant tasks, see Ant Task Reference in WebLogic Web Services: Reference.

**wspackage (deprecated)**

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.

Web Services are now a Java EE standard, which has resulted in many changes between 8.1 and 9.0 WebLogic Web Services. For information about what has changed between 8.1 and 9.0 Web Services, and what has been deprecated, see Differences Between 8.1 and 9.0 WebLogic Web Services in Programming Web Services for Weblogic Server.

For a complete list of Web Services Ant tasks, see Ant Task Reference in WebLogic Web Services: Reference.
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.

Oracle recommends using `java weblogic.Server` primarily for initial development but not as a standard mechanism for starting production systems for the following reasons:

- `java weblogic.Server` will not function if you select a product directory outside of the BEA home directory.
- When executing `java weblogic.Server`, patches will not be recognized by the WebLogic Server run time.

This section describes the following:

- “Required Environment and Syntax for weblogic.Server” on page 3-2
- “Default Behavior” on page 3-3
- “weblogic.Server Configuration Options” on page 3-5
- “Using the weblogic.Server Command Line to Start a Server Instance” on page 3-26
- “Using the weblogic.Server Command Line to Create a Domain” on page 3-27
- “Verifying Attribute Values That Are Set on the Command Line” on page 3-29
For information about using scripts to start an instance of WebLogic Server, see Starting an Administration Server with a Startup Script and Starting Managed Servers With a Startup Script in Managing Server Startup and Shutdown.

For information about using the Node Manager to start an instance of WebLogic Server, see Using Node Manager to Control Servers in the Node Manager Administrator’s 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 *Installation Guide*.

2. If desired, modify the `CLASSPATH` environment variable, as described in “Modifying the Classpath” on page 3-2.

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/platform/suppconfigs/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**

After installation, 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, updating the version of PointBase 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, 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 the JAR file to the PRE_CLASSPATH environment variable.

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

WL_HOME/common/eval/pointbase/lib/pbembedded51.jar and pbclient51.jar

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

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

If you use Log4j logging, include the following file on the classpath:

WL_HOME/server/lib/log4j.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 3-2, when you enter the command java weblogic.Server with no options, WebLogic Server does the following:

1. Looks in the domain_name/config directory for a file named config.xml.

2. If config.xml exists in the domain_name/config directory, WebLogic Server does the following:
   a. If only one server instance is defined in config/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/config.xml:
   - If an Administration Server is defined, it looks for the server with that name.
   - If an Administration Server is not defined, it 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 prompts you to create one. If you respond y, WebLogic Server does the following:
   a. Creates a server configuration named myserver, and persists the configuration in a file named config/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.
      You can configure WebLogic Server to make backup copies of the configuration files. This facilitates recovery in cases where configuration changes need to be reversed or the unlikely case that configuration files become corrupted. For more information, see Configuration File Archiving in Understanding Domain Configuration.
   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 domain_name/security/files named DefaultAuthenticatorInit.ldift, DefaultRoleMapperInit.ldift, and SerializedSystemIni.dat.
      WebLogic Server also encrypts and stores your username and password in a server_name/security/boot.properties file, which enables you to bypass the login prompt during subsequent instantiations of the server. For more information, see Boot Identity Files in Managing Server Startup and Shutdown.
   c. Creates two scripts, bin/startWebLogic.cmd and bin/startWebLogic.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 startWebLogic 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 attributes of a server instance. The following attributes are commonly used when starting a server instance:

- “JVM Parameters” on page 3-5
- “Location of Configuration Data” on page 3-6
- “Options that Override a Server’s Configuration” on page 3-9

WebLogic Server provides other startup options that enable you to temporarily override a server’s saved configuration. For information about these startup options, see “Options that Override a Server’s Configuration” on page 3-9.

Unless you are creating a new domain as described in “Using the weblogic.Server Command Line to Create a Domain” on page 3-27, 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 WebLogic Scripting Tool (WLST) to modify the `config.xml` file. See Creating Domains Using WLST Offline in *WebLogic Scripting Tool*.

For information on verifying the WebLogic Server attribute values that you set, see “Verifying Attribute Values That Are Set on the Command Line” on page 3-29.

**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, see the documentation for your specific JVM. For a list of JVMs that can be used with WebLogic Server, see the Supported Configurations page at http://e-docs.bea.com/platform/suppconfigs/index.html.
**Location of Configuration Data**

All server instances must have access to configuration data. The following table provides options for indicating the location of this data.

### Table 3-1 Frequently Used Options for Setting JVM Parameters

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-Xms</code> and <code>-Xmx</code></td>
<td>Specify the minimum and maximum values (in megabytes) for Java heap memory.</td>
</tr>
<tr>
<td></td>
<td>For example, you might want to start the server with the default allocation of 256 megabytes of Java heap memory to the WebLogic Server. To do so, start the server using the <code>java -Xms256m</code> and <code>-Xmx512m</code> options.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td><code>-classpath</code></td>
<td>The minimum content for this option is described under “Modifying the Classpath” on page 3-2.</td>
</tr>
<tr>
<td></td>
<td>Instead of using this argument, you can use the <code>CLASSPATH</code> environment variable to specify the classpath.</td>
</tr>
<tr>
<td><code>-client</code></td>
<td>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, see the Supported Configurations page at <a href="http://e-docs.bea.com/platform/suppconfigs/index.html">http://e-docs.bea.com/platform/suppconfigs/index.html</a>.</td>
</tr>
<tr>
<td><code>-server</code></td>
<td>To display special characters on Linux browsers, set the JVM’s <code>file.encoding</code> system property to <code>ISO8859_1</code>. For example, <code>java -Dfile.encoding=ISO8859_1</code> <code>weblogic.Server</code>. For a complete listing, see Sun’s “Supported Encodings” page for J2SE 1.6.</td>
</tr>
</tbody>
</table>

To display special characters on Linux browsers, set the JVM’s `file.encoding` system property to `ISO8859_1`. For example, `java -Dfile.encoding=ISO8859_1` `weblogic.Server`. For a complete listing, see Sun’s “Supported Encodings” page for J2SE 1.6.
Table 3-2 Options for Indicating the Location of Configuration Data

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-Dbea.home=bea_home</code></td>
<td>Specifies the location of the BEA home directory, which contains essential information. By default, <code>weblogic.Server</code> determines the location of the BEA home directory based on values in the classpath.</td>
</tr>
<tr>
<td><code>-Dweblogic.RootDirectory=path</code></td>
<td>Specifies the server’s root directory. See A Server’s Root Directory in Understanding Domain Configuration. By default, the root directory is the directory from which you issue the start command.</td>
</tr>
<tr>
<td><code>-Dweblogic.ConfigFile=file_name</code></td>
<td><strong>Note:</strong> This option was removed as of WebLogic Server 9.0. Specifies a configuration file for your domain. The <code>file_name</code> value must see a valid XML file that conforms to the schema at <a href="http://www.bea.com/ns/weblogic/920/domain.xsd">http://www.bea.com/ns/weblogic/920/domain.xsd</a>. 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 <code>-Dweblogic.RootDirectory</code>. The <code>file_name</code> value cannot contain a pathname component. For example, the following value is invalid: <code>-Dweblogic.ConfigFile=c:\mydir\myfile.xml</code> Instead, use the following arguments: <code>-Dweblogic.RootDirectory=c:\mydir</code> <code>-Dweblogic.ConfigFile=myfile.xml</code> If you do not specify this value, the default is <code>config/config.xml</code> in the server’s root directory.</td>
</tr>
</tbody>
</table>
For information on how a Managed Server retrieves its configuration data, see the -Dweblogic.management.server entry in Table 3-3.

The Administration Console does not display values that you set on the command line. For information on verifying the attribute values that you set, see “Verifying Attribute Values That Are Set on the Command Line” on page 3-29.

**Example**

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

Options that Override a Server’s Configuration

In most cases, you do not use startup options to override the configuration that is saved in the
domain’s config.xml file. However, in some extraordinary cases you might need to do so.

Caution: When you use a startup option to override a configuration value, the server instance
uses this value for the duration of its life cycle. Even if you use the Administration
Console, the WebLogic Scripting Tool, or some other utility to change the value in
the configuration, the value will remain overridden until you restart the server without
using the override.

For example, in a production environment, your organization might have a policy against
modifying the domain’s config.xml file, but you need to shut down the Administration Server
and restart it using a temporary listen port. In this case, when you use the weblogic.Server
command to start the Administration Server, you can include the
-Dweblogic.ListenPort=7501 startup option to change the listen port for the current server
session. The server instance initializes its configuration MBeans from the config.xml file but
substitutes 7501 as the value of its listen port. When you subsequently restart the server without
passing the startup option, it will revert to using the value from the config.xml file, 8010. (See
Figure 3-1.)
Figure 3-1  Overriding config.xml Values

1. At startup, servers initialize configuration MBeans with data from the configuration files.

```
config.xml
ListenPort=8010
```

```
java weblogic.Server
-Dweblogic.ListenPort=7501
```

2. Startup options override the values in the configuration files.

The following options temporarily override a server’s configuration:

- “Server Communication” on page 3-10
- “SSL” on page 3-13
- “Security” on page 3-17
- “Message Output and Logging” on page 3-21
- “Clusters” on page 3-26
- “Deployment” on page 3-26
- “Other Server Configuration Options” on page 3-23

**Server Communication**

The following table describes the options for configuring how servers communicate.
Table 3-3 Options for Configuring Server Communication

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-Dweblogic.management.server= {protocol://}Admin-host:port</td>
<td>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. <strong>Note:</strong> 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-wide administration port, port must specify the domain-wide administration port. For more information on configuring a connection to the Administration Server, see Configuring Managed Server Connections to the Administration Server in Managing Server Startup and Shutdown.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-Dweblogic.ListenAddress=host</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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 WLST to modify the config.xml file.</td>
</tr>
<tr>
<td></td>
<td>For more information, see Configure listen addresses in the Administration Console Online Help and Creating Domains Using WLST Offline in WebLogic Scripting Tool.</td>
</tr>
<tr>
<td>-Dweblogic.ListenPort=portnumber</td>
<td>Enables and specifies the plain-text (non-SSL) listen port for the server instance.</td>
</tr>
<tr>
<td></td>
<td>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 WLST to modify the config.xml file.</td>
</tr>
<tr>
<td></td>
<td>The default listen port is 7001.</td>
</tr>
<tr>
<td></td>
<td>For more information, see Configure listen ports in the Administration Console Online Help and Creating Domains Using WLST Offline in WebLogic Scripting Tool.</td>
</tr>
<tr>
<td>-Dweblogic.ssl.ListenPort=portnumber</td>
<td>Enables and specifies the port at which this WebLogic Server instance listens for SSL connection requests.</td>
</tr>
<tr>
<td></td>
<td>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 WLST to modify the config.xml file.</td>
</tr>
<tr>
<td></td>
<td>The default SSL listen port is 7002.</td>
</tr>
<tr>
<td></td>
<td>For more information, see Configure listen ports in the Administration Console Online Help and Creating Domains Using WLST Offline in WebLogic Scripting Tool.</td>
</tr>
</tbody>
</table>
The Administration Console does not display values that you set on the command line. For information on verifying the attribute values that you set, see “Verifying Attribute Values That Are Set on the Command Line” on page 3-29.

SSL

Each Weblogic Server instance uses an instance of
weblogic.management.configuration.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.SSLIgnoreHostnameVerification 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>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-Dweblogic.security.SSL.ignoreHostnameVerification=true</code></td>
<td>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 <code>config.xml</code> file, the server instance does not verify host names when it is in the role of SSL client. <strong>Note:</strong> Oracle 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 <code>config.xml</code> file. The override applies to the current server instantiation; it does not modify the value in the <code>config.xml</code> file. Use the Administration Console or WLST to modify the <code>config.xml</code> file. For more information, see <strong>Using Hostname Verification</strong> in <strong>Securing WebLogic Server</strong>.</td>
</tr>
<tr>
<td><code>-Dweblogic.security.SSL.HostnameVerifier=hostnameverifierimplementation</code></td>
<td>Specifies the name of a custom Host Name Verifier class. The class must implement the <code>weblogic.security.SSL.HostnameVerifier</code> interface.</td>
</tr>
</tbody>
</table>
### Table 3-4 Options for Configuring SSL

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-Dweblogic.security.SSL.nojce=true</code></td>
<td>Specifies server uses a FIPS-compliant (FIPS 140-2) crypto module for SSL.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> To start a server instance so that it uses a FIPS-compliant (FIPS 140-2) crypto module in its SSL implementation, you must also ensure that <em>jsafeFIPS.jar</em> is added to the <em>PRE_CLASSPATH</em> variable in the server start script (for example, <code>startWebLogic.cmd/sh</code>).</td>
</tr>
<tr>
<td><code>-Dweblogic.security.SSL.sessionCache.ttl=</code></td>
<td>Modifies the default server-session time-to-live for SSL session caching.</td>
</tr>
<tr>
<td><code>sessionCacheTimeToLive</code></td>
<td>The <em>sessionCacheTimeToLive</em> 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 <code>-Dweblogic.security.SSL.sessionCache.ttl</code> in the server startup script.</td>
</tr>
<tr>
<td></td>
<td><strong>For sessionCache.ttl:</strong></td>
</tr>
<tr>
<td></td>
<td>• The minimum value is 1.</td>
</tr>
<tr>
<td></td>
<td>• The maximum value is <code>Integer.MAX_VALUE</code></td>
</tr>
<tr>
<td></td>
<td>• The default value is <code>90000</code></td>
</tr>
</tbody>
</table>

---

weblogic.Server Configuration Options
The Administration Console does not display values that you set on the command line. For information on verifying the attribute values that you set, see “Verifying Attribute Values That Are Set on the Command Line” on page 3-29.

### 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.

### Table 3-4 Options for Configuring SSL

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| `-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`.  
  Oracle recommends that you do not use the demonstration certificate authorities in any type of production deployment.  
  For more information, see Configuring SSL in the Securing Weblogic Server. |
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:

   ```java
   setEnabled()
   ```

   To enable SSL for a server instance named MedRecServer, use the following command when you start MedRecServer:

   ```java
   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, see “Verifying Attribute Values That Are Set on the Command Line” on page 3-29.

**Security**

The following table describes the options for configuring general security parameters.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-Dweblogic.management.username=username</code></td>
<td>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, see Users, Groups, and Security Roles in Securing WebLogic Resources. 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, see Boot Identity Files in Managing Server Startup and Shutdown.</td>
</tr>
<tr>
<td><code>-Dweblogic.management.password=password</code></td>
<td>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, see Boot Identity Files in Managing Server Startup and Shutdown. Note: If you supply the password, but no username, you will be prompted for both the username and the password.</td>
</tr>
</tbody>
</table>
Table 3-5 Options for General Security Parameters

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-Dweblogic.system. StoreBootIdentity=true</td>
<td>Creates a <code>boot.properties</code> 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. For more information, see Specifying User Credentials When Starting a Server with the Node Manager in Managing Server Startup and Shutdown. Oracle recommends that you do not add this argument to a startup script. Instead, use it only when you want to create a <code>boot.properties</code> file. For more information, see Boot Identity Files in Managing Server Startup and Shutdown.</td>
</tr>
<tr>
<td>-Dweblogic.system. BootIdentityFile=filename</td>
<td>Specifies a boot identity file that contains a username and password. The <code>filename</code> value must be the fully qualified pathname of a valid boot identity file. For example: <code>-Dweblogic.system.BootIdentityFile=WL_HOME\mydomain\servers\myserver\security\boot.properties</code> If you do not specify a filename, a server instance or the <code>weblogic.Admin SHUTDOWN</code> and <code>FORCESHUTDOWN</code> commands use the <code>boot.properties</code> 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 <code>weblogic.Admin SHUTDOWN</code> and <code>FORCESHUTDOWN</code> commands, you must use the <code>-username</code> and <code>-password</code> arguments to provide user credentials. Note: The <code>weblogic.Admin</code> utility is deprecated in WebLogic Server 9.0. Oracle recommends that you use the WebLogic Scripting Tool (WLST) for equivalent functionality such as <code>SHUTDOWN</code> and <code>FORCESHUTDOWN</code>. For more information on using these commands, see “Life Cycle Commands” in the WLST Command and Variable Reference.</td>
</tr>
</tbody>
</table>
Table 3-5 Options for General Security Parameters

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-Dweblogic.system.RemoveBootIdentity=true</code></td>
<td>Removes the boot identity file after a server starts.</td>
</tr>
<tr>
<td><code>-Dweblogic.security.anonymous</code></td>
<td>Assigns a user ID to anonymous users. By default, all anonymous users are identified with the string <code>&lt;anonymous&gt;</code>.</td>
</tr>
<tr>
<td><code>UserName=name</code></td>
<td>To emulate the security behavior of WebLogic Server 6.x, specify <code>guest</code> for the <code>name</code> value and create a user named <code>guest</code> in your security realm.</td>
</tr>
<tr>
<td></td>
<td>For more information, see “Users, Groups, an Security Roles” in Securing WebLogic Resources.</td>
</tr>
<tr>
<td><code>-Djava.security.manager</code></td>
<td>Standard Java EE options that enable the Java security manager and specify a filename (using a relative or fully-qualified pathname) that contains Java 2 security policies.</td>
</tr>
<tr>
<td><code>-Djava.security.policy[=]=filename</code></td>
<td>To use the WebLogic Server sample policy file, specify <code>WL_HOME\server\lib\weblogic.policy</code>.</td>
</tr>
<tr>
<td></td>
<td>Using <code>-Djava.security.policy==filename</code> (note the double equal sign <code>==</code>) causes the policy file to override any default security policy. This causes WebLogic Server to ignore any policy files that are used for servlet and EJB authorization when JACC is enabled. A single equal sign <code>=</code> causes the policy file to be appended to an existing security policy.</td>
</tr>
<tr>
<td></td>
<td>For more information, see “Using the Java Security Manager to Protect WebLogic Resources” in the Programming WebLogic Security guide.</td>
</tr>
<tr>
<td><code>-Dweblogic.security.fullyDelegateAuthorization=true</code></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Use this option when starting WebLogic Server to override this problem.</td>
</tr>
<tr>
<td></td>
<td>This startup option does not work with EJBs or EJB methods that use <code>&lt;unchecked&gt;</code> or <code>&lt;restricted&gt;</code> tags or Web applications that do not have a role-name specified in the <code>&lt;auth-constraint&gt;</code> tag.</td>
</tr>
</tbody>
</table>
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 domain_name→Security→General page 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, see Users, Groups, an Security Roles in Securing WebLogic Resources.

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 -1 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.
The Administration Console does not display values that you set on the command line. For information on verifying the attribute values that you set, see “Verifying Attribute Values That Are Set on the Command Line” on page 3-29.

**Message Output and Logging**

The following table describes options for configuring a server instance’s message output.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-Djava.security.manager</td>
<td>Defining these five system properties is required to enable the use of the JACC provider in the security realm. When these providers are in use, the JACC handles authorization decisions for the EJB and Servlet containers for external applications. Any other authorization decisions for internal applications are handled by the authorization in the WebLogic Security framework. JACC authorization requires the use of J2SE security and therefore requires that WebLogic Server be booted with a Java EE security manager and a policy file (specified by the server startup properties, java.security.manager and java.security.policy). For more information, see “Using the Java Security Manager to Protect WebLogic Resources” in Programming WebLogic Security. The WebLogic JACC implementation expects that the policy object is the default sun.security.provider.PolicyFile class. When starting, WebLogic Server attempts to locate and instantiate the classes specified by the JACC startup properties and fails if it cannot find or instantiate them (if, for example, the files specified by the startup properties are not valid classes).</td>
</tr>
<tr>
<td>-Djava.security.policy= &lt;insert the location of your policy file here&gt;</td>
<td></td>
</tr>
<tr>
<td>-Djavax.security.jacc.PolicyConfigurationFactory.provider= weblogic.security.jacc.simple provider.PolicyConfigurationFactoryImpl</td>
<td></td>
</tr>
<tr>
<td>-Djavax.security.jacc.policy.provider= weblogic.security.jacc.simple provider.SimpleJACCProvider</td>
<td></td>
</tr>
<tr>
<td>-Dweblogic.security.jacc.RoleMapperFactory.provider= weblogic.security.jacc.simple provider.RoleMapperFactoryImpl</td>
<td></td>
</tr>
<tr>
<td>-Dweblogic.security.ldap.maxSize=&lt;max bytes&gt;</td>
<td>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.</td>
</tr>
<tr>
<td>-Dweblogic.security.ldap.changeLogThreshold=&lt;number of entries&gt;</td>
<td>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.</td>
</tr>
</tbody>
</table>
The Administration Console does not display values that you set on the command line. For information on verifying the attribute values that you set, see “Verifying Attribute Values That Are Set on the Command Line” on page 3-29.

### 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.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-Dweblogic.Stdout=&quot;filename&quot;</code></td>
<td>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, see Redirect JVM output in the Administration Console Online Help.</td>
</tr>
<tr>
<td><code>-Dweblogic.Stderr=&quot;filename&quot;</code></td>
<td>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, see Redirecting JVM output in Configuring Log Files and Filtering Log Messages.</td>
</tr>
<tr>
<td><code>-Dweblogic.AdministrationMBeanAuditingEnabled=</code></td>
<td>Determines whether the Administration Server emits configuration auditing log messages when a user changes the configuration or invokes management operations on any resource within a domain. By default, the Administration Server does not emit configuration auditing messages. See “Enable configuration auditing” in the Administration Console Online Help.</td>
</tr>
</tbody>
</table>

Table 3-6: Options for Configuring Message Output
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, see “Verifying Attribute Values That Are Set on the Command Line” on page 3-29.

**Other Server Configuration Options**

The following table describes options for configuring additional attributes of a server instance.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-DserverType=wlx</td>
<td>Starts the server without starting EJB, JCA, and JMS services.</td>
</tr>
<tr>
<td>-Dweblogic.Name=server</td>
<td>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.</td>
</tr>
</tbody>
</table>
This attribute is **deprecated** in WebLogic Server 9.0.

Determines whether a server starts in production mode.

A `true` value prevents a WebLogic Server from automatically deploying and updating applications that are in the `domain_name/autodeploy` directory.

If you do not specify this option, the assumed value is `false`.

To enable production mode, you can use WLST to set `DomainMBean.isProductionModeEnabled` to `true`, or use the Administration Console. See Change to production mode in the Administration Console Online Help.

**Note:** It is recommended that you enable production mode via the Administration Console or in `config.xml`. You should only enable production mode from the command line on the Administration Server.

**Note:** It is important to note that when `ProductionModeEnabled` is set from the command line on the Administration Server, this value is propagated to all managed servers.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-Dweblogic.ProductionModeEnabled</code></td>
<td>Determines whether a server starts in production mode. A <code>true</code> value prevents a WebLogic Server from automatically deploying and updating applications that are in the <code>domain_name/autodeploy</code> directory.</td>
</tr>
<tr>
<td></td>
<td>If you do not specify this option, the assumed value is <code>false</code>.</td>
</tr>
<tr>
<td></td>
<td>To enable production mode, you can use WLST to set <code>DomainMBean.isProductionModeEnabled</code> to <code>true</code>, or use the Administration Console. See Change to production mode in the Administration Console Online Help.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> It is recommended that you enable production mode via the Administration Console or in <code>config.xml</code>. You should only enable production mode from the command line on the Administration Server.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> It is important to note that when <code>ProductionModeEnabled</code> is set from the command line on the Administration Server, this value is propagated to all managed servers.</td>
</tr>
<tr>
<td><code>-Dweblogic.management.startupMode</code></td>
<td><code>STANDBY</code> starts a server and places it in the <code>STANDBY</code> state. To use this startup argument, the domain must be configured to use the domain-wide administration port.</td>
</tr>
<tr>
<td></td>
<td>For information about administration ports, see “Administration Port and Administrative Channel” in Configuring WebLogic Server Environments and Configure the domain-wide administration port in the Administration Console Online Help.</td>
</tr>
<tr>
<td></td>
<td><code>ADMIN</code> starts a server and places it in the <code>ADMIN</code> state. See <code>ADMIN</code> state in Managing Server Startup and Shutdown.</td>
</tr>
<tr>
<td></td>
<td>Specifying the startup mode startup option overrides any startup mode setting in the <code>config.xml</code> file. The override applies to the current server instantiation; it does not modify the value in the <code>config.xml</code> file. Use the Administration Console or WLST to modify the <code>config.xml</code> file.</td>
</tr>
<tr>
<td></td>
<td>If you do not specify this value (either on the command line or in <code>config.xml</code>), the default is to start in the <code>RUNNING</code> state.</td>
</tr>
</tbody>
</table>
The Administration Console does not display values that you set on the command line. For information on verifying the attribute values that you set, see “Verifying Attribute Values That Are Set on the Command Line” on page 3-29.

### Table 3-7 Options for Configuring Server Attributes

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-Dweblogic.apache.xerces.maxentityrefs=numerical-value</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>-Dweblogic.jsp.windows.casesensitive=true</code></td>
<td>Causes the JSP compiler on Windows systems to preserve case when it creates output files names. See “Using the WebLogic JSP Compiler” in <em>Developing Web Applications, Servlets, and JSPs for WebLogic Server</em>.</td>
</tr>
<tr>
<td><code>-Dweblogic.servlet.optimisticSerialization=true</code></td>
<td>When optimistic-serialization is turned on, WebLogic Server does not serialize-deserialize context and request attributes upon <code>getAttribute(name)</code> 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 <code>getAttribute(name)</code> to avoid the possibility of <code>ClassCastException</code>. The optimistic-serialization value can also be specified at domain level in the <code>WebAppContainerMBean</code>, which applies for all Web applications. The value in <code>weblogic.xml</code>, if specified, overrides the domain level value. The default value is false.</td>
</tr>
</tbody>
</table>
Clusters

The following table describes options for configuring additional attributes of a cluster.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-Dweblogic.cluster.multicastAddress</td>
<td>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.</td>
</tr>
<tr>
<td><strong>Note:</strong></td>
<td>The Administration Console does not display values that you set on the command line. For information on verifying the attribute values that you set, see “Verifying Attribute Values That Are Set on the Command Line” on page 3-29.</td>
</tr>
<tr>
<td></td>
<td>Regardless of how you set the Multicast Address, all servers in a cluster must communicate at the same Multicast Address.</td>
</tr>
</tbody>
</table>

Deployment

The following table describes options for configuring additional attributes for deployment.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-Dweblogic.deployment.IgnorePrepareStateFailures=true</td>
<td>Overrides the default deployment behavior by allowing a server to transition to Running even with static deployment Prepare failures.</td>
</tr>
<tr>
<td><strong>Note:</strong></td>
<td>This server level flag may cause inconsistent deployment behavior within clusters, such as issues with HttpSessionReplication or SFSB replication.</td>
</tr>
</tbody>
</table>

Using the weblogic.Server Command Line to Start a Server Instance

A simple way to start a server instance is as follows:
Using the weblogic.Server Command Line to Create a Domain

1. In a command shell, set up the required environment variables by running the following script:
   
   \texttt{WL\_HOME/server/bin/setWLSEnv.cmd} \textbf{(on Windows)}
   
   \texttt{WL\_HOME/server/bin/setWLSEnv.sh} \textbf{(on UNIX)}
   
   where \texttt{WL\_HOME} is the directory in which you installed the WebLogic Server software.

2. In the command shell, change to the root of the domain directory, usually \texttt{BEA\_HOME/user\_projects/domains/DOMAIN\_NAME}. For example, change to the \texttt{WL\_HOME/samples/domains/medrec} directory.

3. To start an Administration Server, enter the following command:
   
   \texttt{java weblogic.Server}
   
   \textbf{Note:} The password you use must be a string of at least 8 case-sensitive characters. The space character is not supported. For more information, see “Configure an Administrator Username and Password” in Creating WebLogic Domains Using the Configuration Wizard.

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:
   
   \texttt{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:
   
   \texttt{java -Dweblogic.Name=MedRecManagedServer -Dweblogic.management.server=localhost:7011 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 3-3, 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=7001
   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 then 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, see Configure the listen addresses in the Administration Console Online Help.
   - The Administration Server listens on port 7001.

Entering the `weblogic.Server` command as described in this section creates the following files:

- config.xml
- `DefaultAuthenticatorInit.ldift`, `DefaultRoleMapperInit.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
Verifying Attribute Values That Are Set on the Command Line

start the server. For more information, see Boot Identity Files in Managing Server Startup and Shutdown.

- startWebLogic.cmd and startWebLogic.sh, that you can use to start subsequent instantiations of the server.

Note: Invoking weblogic.Server in an empty directory results in implicit domain creation which uses the same configuration process as WLST offline and the Configuration Wizard and thus ensures that you always see uniform domains. As a result, implicitly creating a domain in an empty directory using weblogic.Server may take around 15

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 WLST as follows:

1. Follow “Main Steps for Using WLST” which includes “Setting Up Your Environment” and “Invoking WLST” in WebLogic Scripting Tool.
   
   java weblogic.WLST

2. Start a WebLogic Server instance (see Starting and Stopping Servers) and connect WLST to the server using the connect command. For detailed information about the connect command, see “connect” in the WLST Command and Variable Reference.
   
   wls://(offline)> connect('username','password','t3s://localhost:7002')
   Connecting to weblogic server instance running at t3s://localhost:7002 as username weblogic ...
   wls:/mydomain/serverConfig>

3. For example, to determine the multicast address that a cluster member is using, connect WLST to that server instance and enter the following commands:

   wls:/mydomain/serverConfig> cd('Clusters/cluster_name')
   wls:/mydomain/serverConfig/Clusters/mycluster>
   cmo.getMulticastAddress()
   '239.192.0.0'

4. To determine the severity level of messages that the server instance prints to standard out, connect WLST to that server instance and enter the following commands:
wls:/mydomain/serverConfig> cd('Servers/server_name/Log/server_name')
wls:/mydomain/serverConfig/Servers/myserver/Log/myserver> cmo.getStdoutSeverity()
'Notice'

For more information on using WLST, see WebLogic Scripting Tool. For more information about configuration MBeans, see Understanding WebLogic Server MBeans in Developing Custom Management Utilities with JMX.
WebLogic SNMP Agent Command-Line Reference (Deprecated)

Note: The command-line utility that this document describes is deprecated in WebLogic Server 10.0. Instead, use the command-line utility that is described in WebLogic SNMP Command-Line Utility in WebLogic SNMP Management Guide.

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 (managed objects), 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 managed objects from all Managed Servers within a domain.

The WebLogic SNMP agent provides a command-line interface that lets you:

- Retrieve WebLogic Server managed objects.
- Generate and receive WebLogic Server traps for testing purposes.

The following sections describe working with the WebLogic SNMP agent through its command-line interface:

- “Required Environment for the SNMP Command-Line Interface” on page 4-2
- “Syntax and Common Arguments for the SNMP Command-Line Interface” on page 4-2
- “Commands for Retrieving WebLogic Server Managed Objects” on page 4-4
- “Commands for Testing Traps” on page 4-9

For more information about using SNMP with WebLogic Server, see:
Required Environment for the SNMP Command-Line Interface

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 Installation Guide.

2. If you want to retrieve WebLogic Server managed objects, enable the WebLogic SNMP agent as described in Use SNMP to Monitor WebLogic Server in the Administration Console Online Help.

3. Open a command prompt (shell) and invoke the following script:

   ```bash
   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.

Syntax and Common Arguments for the SNMP Command-Line Interface

All WebLogic SNMP agent commands take the following form:

```
java command-name arguments
```

Table 4-1 describes arguments that are common to most WebLogic SNMP agent commands.
Syntax and Common Arguments for the SNMP Command-Line Interface

Table 4-1  Common Command Line Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>-d</td>
<td>Includes debugging information and packet dumps in the command output.</td>
</tr>
<tr>
<td>-v {v1</td>
<td>v2}</td>
</tr>
<tr>
<td>-c snmpCommunity[@server_name]</td>
<td>The community name that you set for the WebLogic SNMP agent and optionally specifies the server instance that hosts the objects with which you want to interact. To request a managed 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 Create SNMP agents in the Administration Console Online Help). To request a managed object on a single Managed Server, specify: snmpCommunity@server_name where server_name is the name of the Managed Server. To request a managed object for all server instances in a domain, specify a community string with the following form: snmpCommunity@domain_name where domain_name is the name of the WebLogic Server domain. If you do not specify a value for this argument, the command assumes -c public, which uses the default community name, and assumes that the specified managed object is on the Administration Server.</td>
</tr>
<tr>
<td>-p snmpPort</td>
<td>The port number on which the WebLogic SNMP agent listens for requests. If you do not specify a value, the command assumes -p 161.</td>
</tr>
<tr>
<td>-t timeout</td>
<td>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.</td>
</tr>
</tbody>
</table>
Table 4-1  Common Command Line Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
</table>
| `-r retries` | 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` | 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 WebLogic Server Managed Objects

Table 4-2 is an overview of commands that retrieve WebLogic Server managed objects and object instances.

Table 4-2  Overview of Commands for Retrieving Data from WebLogic Server Managed Objects

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
</table>
| `snmpwalk`    | Returns all managed objects and instances that are below a specified node in the MIB.  
See “`snmpwalk`” on page 4-4. |
| `snmpgetnext` | Returns the managed object or instance that immediately follows an OID that you specify.  
See “`snmpgetnext`” on page 4-6. |
| `snmpget`     | Returns managed object instances that correspond to one or more OIDs.  
See “`snmpget`” on page 4-8. |

`snmpwalk`

Returns all managed objects or instances that are below a specified node in the MIB.  
If you specify the OID for a tabular object, the command returns all of its object instances along with all related (child) objects and instances.
Syntax

```
java snmpwalk [-d] [-v (v1,v2)] [-c snmpCommunity] [-p snmpPort]
               [-t timeout] [-r retries] host OID
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>OID</em></td>
<td>The object ID of the node from which you want to retrieve a set of child objects and instances. 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.</td>
</tr>
</tbody>
</table>

Example

The following example retrieves the names of all applications that have been deployed on the Administration Server. The managed object for an application name is `applicationRuntimeName`, which is a child of the `applicationRuntimeTable` object. (See WebLogic Server SNMP MIB Reference.)

```
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.

```
Object ID:
```
```
.1.3.6.1.4.1.140.625.105.1.15.32.49.102.98.97.100.97.102.99.57.48.50.102.48.98.53.54.100.100.49.54.50.54.99.54.49.97.97.97.98.53.100.97
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.

java snmpwalk -c public@medrec localhost .1.3.6.1.4.1.140.625.105.1.15

The following example retrieves the name of all applications that have been deployed on a Managed Server named MS1.

java snmpwalk -c public@MS1 localhost .1.3.6.1.4.1.140.625.105.1.15

**snmpgetnext**

Returns a description of the managed object or object instance that immediately follows one or more OIDs that you specify. If you specify a tabular object, this command returns the first child managed object. If you specify a scalar object, this command returns the first instance of the object.

Instead of the recursive listing that the `snmpwalk` command provides, this command returns the description of only one managed object or instance 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.

**Syntax**

```
java snmpgetnext [-d] [-v (v1,v2)] [-c snmpCommunity] [-p snmpPort]
    [-t timeout] [-r retries] host OID [OID]...
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>OID [OID]...</code></td>
<td>One or more object IDs. Use a space to delimit multiple OIDs. You can specify OIDs for objects or instances. 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.</td>
</tr>
</tbody>
</table>

**Example**

The following example retrieves the name of an application that has been deployed on the Administration Server. The managed object for an application name is `applicationRuntimeName`, which is a scalar object and is a child of the `applicationRuntimeTable` object. (See WebLogic Server SNMP MIB Reference.)
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: .1.3.6.1.4.1.140.625.105.1.15.32.49.102.98.97.100.97.102.99.57.48.50.102.48.98.53.54.100.100.49.54.50.54.99.49.97.97.98.95.100.97
STRING: MedRecServer_uddiexplorer

To determine whether there are additional applications deployed on the Administration Server, you can use the output of the `snmpgetnext` command as input for an additional `snmpgetnext` command:

java snmpgetnext localhost .1.3.6.1.4.1.140.625.105.1.15.32.49.102.98.97.100.97.102.99.57.48.50.102.48.98.53.54.100.100.49.54.50.54.99.49.97.97.98.95.100.97

The command returns output similar to the following:

Response PDU received from /127.0.0.1, community: public
Object ID: .1.3.6.1.4.1.140.625.105.1.15.32.54.98.49.101.57.56.54.98.98.50.102.48.98.53.54.100.100.49.54.50.54.99.49.97.97.98.95.100.97
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, which is the name of an application, and `jdbcConnectionPoolRuntimeName`, which is the name of a JDBC connection pool.

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: .1.3.6.1.4.1.140.625.105.1.15.32.49.102.98.97.100.97.102.99.57.48.50.102.48.98.53.54.100.100.49.54.50.54.99.49.97.97.98.95.100.97
STRING: MedRecServer_uddiexplorer
Object ID:
.1.3.6.1.4.1.140.625.190.1.15.32.53.54.49.48.50.55.52.57.57.49.99.102
.55.48.98.53.50.54.100.48.100.53.52.56.49.57.49.49.99.99.99
STRING: MedRecPool-PointBase

snmpget

Retrieves the value of one or more object instances. This command does not accept OIDs for managed objects.

Syntax

java snmpget [-d] [-v (v1,v2)] [-c snmpCommunity] [-p snmpPort]
[-t timeout] [-r retries] host object-instance-OID
[object-instance-OID]...

Table 4-5  snmpget Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>object-instance-OID</td>
<td>The object ID of an object instance. This command does not accept OIDs for managed objects.</td>
</tr>
<tr>
<td>[object-instance-OID]...</td>
<td>Start the value with '.'; otherwise, references are assumed to be relative to the standard MIB, not the WebLogic Server MIB.</td>
</tr>
</tbody>
</table>

Example

The following example retrieves the serverRuntimeState and serverRuntimeListenPort managed object instances for the Administration Server. Both of these objects are children of the serverRuntimeTable object. (See WebLogic Server SNMP MIB Reference.)

java snmpget localhost
.1.3.6.1.4.1.140.625.360.1.60.32.102.100.48.98.101.102.100.99.102.52.98.97.48.49.102.57.53.51.50.100.102.53.55.97.101.52.56.99.99.97.99
.1.3.6.1.4.1.140.625.360.1.35.32.102.100.48.98.101.102.100.99.102.52.98.97.48.49.102.57.53.51.50.100.102.53.55.97.101.52.56.99.99.97.99

If you invoke this command from a computer that is running the example MedRecServer, the command returns output similar to the following:
Commands for Testing Traps

Table 4-6 is an overview of commands that generate and receive traps for testing purposes.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>snmptrapd</td>
<td>Starts a daemon that receives traps and prints information about the trap.</td>
</tr>
<tr>
<td></td>
<td>See “snmptrapd” on page 4-9.</td>
</tr>
<tr>
<td>snmpv1trap</td>
<td>Constructs an SNMPv1 trap and distributes it to the SNMP manager or trap</td>
</tr>
<tr>
<td></td>
<td>daemon that is running on the specified host and listening on the specified</td>
</tr>
<tr>
<td></td>
<td>port number.</td>
</tr>
<tr>
<td></td>
<td>See “snmpv1trap” on page 4-10.</td>
</tr>
</tbody>
</table>

**snmptrapd**

Starts a daemon that receives traps and prints information about the trap.

**Syntax**

```java
java snmptrapd [-d] [-c snmpCommunity] [-p TrapDestinationPort]
```
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.

**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.

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 **OIDs for WebLogic Server 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] ...
```
## Table 4-8 snmpv1trap Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-c snmpCommunity</code></td>
<td>A community name for the trap. SNMP managers (or the trap daemon) can access the trap only if they are configured to use this community name. If you do not specify a value, the command assumes <code>-c public</code>.</td>
</tr>
<tr>
<td><code>-p TrapDestinationPort</code></td>
<td>The port number on which the SNMP manager or trap daemon is listening. If you do not specify a value, the command assumes <code>-p 162</code>.</td>
</tr>
<tr>
<td>TrapDestinationHost</td>
<td>The DNS name or IP address of the computer that hosts the SNMP manager or trap daemon.</td>
</tr>
<tr>
<td><code>.1.3.6.1.4.140.625</code></td>
<td>The value of the trap’s enterprise field, which contains the beginning portion of the OID for all WebLogic Server traps.</td>
</tr>
<tr>
<td><code>agent-addr</code></td>
<td>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 <code>snmpv1trap</code> command to generate a trap, you can specify any valid DNS name or IP address.</td>
</tr>
<tr>
<td><code>generic-trap</code></td>
<td>The value of the trap’s generic trap type field. For a list of valid values, refer to OIDs for WebLogic Server Notifications in the WebLogic SNMP Management Guide.</td>
</tr>
<tr>
<td><code>specific-trap</code></td>
<td>The value of the trap’s specific trap type field. For a list of valid values, refer to OIDs for WebLogic Server Notifications in the WebLogic SNMP Management Guide.</td>
</tr>
</tbody>
</table>
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 .1.3.6.1.4.1.140.625.100.5 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
```
Example: Using \texttt{snmpv1trap} to Send Traps to the Trap Daemon

To use the \texttt{snmpv1trap} command to generate WebLogic Server traps and receive them through the trap daemon:

1. Open a command prompt (shell) and invoke the following script:

   \texttt{WL\_HOME\bin\setWLSEnv.sh} (or \texttt{setWLSEnv.cmd} on Windows)

   where \texttt{WL\_HOME} is the directory in which you installed WebLogic Server.

2. To start the trap daemon, enter the following command:

   \texttt{java snmptrapd}

3. Open another shell and invoke the following script:

   \texttt{WL\_HOME\bin\setWLSEnv.sh} (or \texttt{setWLSEnv.cmd} on Windows)

4. To generate a trap, enter the following command:

   \texttt{java snmpv1trap localhost \hphantom{1}1.3.6.1.4.140.625 \hphantom{1}localhost 6 60 1000}

The \texttt{snmpv1trap} command generates a \texttt{serverStart} trap and broadcasts it through port 162.

In the shell in which the trap daemon is running, the daemon prints the following:

\begin{verbatim}
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
\end{verbatim}

VARBINDS:
Example: Using the WebLogic SNMP Agent to Send Traps to the Trap Daemon

To use WebLogic SNMP agent to generate WebLogic Server traps and receive them through the trap daemon:

1. Start the Administration Server for a domain and enable the SNMP agent.
   See Create SNMP agents in the Administration Console Online Help.

2. Create a trap destination to represent the trap daemon. Configure the trap destination to use port 165. Keep all other default settings that the Administration Console presents.
   See Create Trap Destinations in the Administration Console Online Help.

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.

4. To start the trap daemon, enter the following command:
   java snmptrapd -p 165

5. Restart the Administration Server.
When the Administration Server starts, the SNMP agent generates a serverStart trap and broadcasts it through port 165.

   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: 65
   Time: 1000
   VARBINDS: