BEA WebLogic Server®

WebLogic Scripting Tool

Version 9.1
Revised: December 16, 2005

Copyright

Restricted Rights Legend

Trademarks and Service Marks

Contents

1. Introduction and Roadmap

2. Using the WebLogic Scripting Tool

3. Creating and Configuring WebLogic Domains Using WLST Offline

4. Navigating and Editing MBeans

5. Managing Servers and Server Life Cycle

6. Automating WebLogic Server Administration Tasks

WLST Command and Variable Reference

WLST Online and Offline Command Summary

WLST Deployment Objects

FAQs: WLST

CHAPTER 1

Introduction and Roadmap

This section describes the contents and organization of this guide—*WebLogic Scripting Tool*.

Document Scope and Audience

This document describes the BEA WebLogic Scripting Tool (WLST). It explains how you use the WLST command-line scripting interface to configure, manage, and persist changes to WebLogic Server® instances and domains, and monitor and manage server runtime events.

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

Guide to This Document

This document is organized as follows:

- This chapter, "Introduction and Roadmap," introduces the organization of this guide and lists related documentation.

- Chapter 2, "Using the WebLogic Scripting Tool," describes how the scripting tool works, its modes of operation, and the basic steps for invoking it.
- Chapter 3, "Creating and Configuring WebLogic Domains Using WLST Offline," describes how to create a new domain or update an existing domain without connecting to a running WebLogic Server (that is, using WLST offline)—supporting the same functionality as the Configuration Wizard.
- Chapter 4, "Navigating and Editing MBeans," describes how to retrieve domain configuration and runtime information, and edit configuration or custom MBeans.
- Chapter 5, "Managing Servers and Server Life Cycle," describes using WLST to start and stop WebLogic Server instances and to monitor and manage the server life cycle.
- Chapter 6, "Automating WebLogic Server Administration Tasks," describes using scripts to automate the creation and management of domains, servers, and resources.
- Appendix A, "WLST Command and Variable Reference," provides detailed descriptions for each of the WLST commands and variables.
- Appendix B, "WLST Online and Offline Command Summary," summarizes WLST commands alphabetically and by online/offline usage.
- Appendix C, "WLST Deployment Objects," describes WLST deployment objects that you can use to update a deployment plan or access information about the current deployment activity.
- Appendix D, "FAQs: WLST," provides a list of common questions and answers.

Related Documentation

- "Using Ant Tasks to Configure and Use a WebLogic Server Domain" in *Developing Applications With WebLogic Server*
- *Administration Console Online Help*
- *Creating WebLogic Domains Using the Configuration Wizard*
- *Developing Custom Management Utilities with JMX*
- *WebLogic SNMP Agent Command-Line Reference*

WLST Sample Scripts

The following sections describe the WLST online and offline sample scripts that you can run or use as templates for creating additional scripts. For information about running scripts, see "Running Scripts" on page 2-10.

WLST Online Sample Scripts

The WLST online sample scripts demonstrate how to perform administrative tasks and initiate WebLogic Server configuration changes while connected to a running server. WLST online scripts are located in the following directory: *SAMPLES_HOME*\server\examples\src\examples\wlst\online, where *SAMPLES_HOME* refers to the main examples directory of your WebLogic Server installation, such as c:*beahome*\weblogic91\samples.

Table 1-1 summarizes WLST online sample scripts.

Table 1-1 WLST Online Sample Scripts

WLST Sample Script	Description
cluster_creation.py	Connects WLST to an Administration Server, starts an edit session, and creates 10 Managed Servers. It then creates two clusters, assigns servers to each cluster, and disconnects WLST from the server.
cluster_deletion.py	Removes the clusters and servers created in cluster_creation.py.
jdbc_data_source_creation.py	Connects WLST to an Administration Server, starts an edit session, and creates a JDBC data source called *myJDBCDataSource*.
jdbc_data_source_deletion.py	Removes the JDBC data source created by jdbc_data_source_creation.py.
configJMSSystemResource.py	Connects WLST to an Administration Server, starts an edit session, creates two JMS Servers, and targets them to the Administration Server. Then creates JMS topics, JMS queues, and JMS templates in a JMS System module. The JMS queues and topics are targeted using sub-deployments.
deleteJMSSystemResource.py	Removes the JMS System module created by configJMSSystemResource.py.

WLST Offline Sample Scripts

The WLST offline sample scripts demonstrate how to create domains using the domain templates that are installed with the software. The WLST offline scripts are located in the following directory: *WL_HOME*\common\templates\scripts\wlst, where *WL_HOME* refers to the top-level installation directory for WebLogic Server.

Table 1-2 summarizes WLST offline sample scripts.

Table 1-2 WLST Offline Sample Script

WLST Sample Script	Description
basicWLSDomain.py	Creates a simple WebLogic domain demonstrating how to open a domain template, create and edit configuration objects, and write the domain configuration information to the specified directory. The sample consists of a single server, representing a typical development environment. This type of configuration is not recommended for production environments. The script uses the Basic WebLogic Server Domain template.
clusterMedRecDomain.py	Creates a single-cluster domain, creating three Managed Servers and assigning them to a cluster. The script uses the Basic WebLogic Server Domain template and extends it using the Avitek Medical Records Sample extension template.
distributedQueues.py	Demonstrates two methods for creating distributed queues. The script uses the Basic WebLogic Server Domain template and extends it using the Avitek Medical Records Sample.
sampleMedRecDomain.py	Creates a domain that defines resources similar to those used in the Avitek MedRec sample. This example does not recreate the MedRec example in its entirety, nor does it deploy any sample applications. This example demonstrates steps similar to those shown in the WLST online script described in "Creating a Sample Domain: Main Steps" on page 6-2, but uses WLST offline. The script uses the Basic WebLogic Server Domain template.

In addition, BEA provides sample scripts to configure WebLogic domain resources using WLST offline and online on the *dev2dev* Web site. For more information, see the *wlst Project Home* at https://wlst.projects.dev2dev.bea.com.

CHAPTER 2

Using the WebLogic Scripting Tool

The following sections describe the WebLogic Scripting Tool:

What is the WebLogic Scripting Tool?

The WebLogic Scripting Tool (WLST) is a command-line scripting interface that system administrators and operators use to monitor and manage WebLogic Server instances and domains. The WLST scripting environment is based on the Java scripting interpreter, Jython. In addition to WebLogic scripting functions, you can use common features of interpreted languages, including local variables, conditional variables, and flow control statements. WebLogic Server developers and administrators can extend the WebLogic scripting language to suit their environmental needs by following the Jython language syntax. See `http://www.jython.org`.

What Does WLST Do?

WLST lets you perform the following tasks:

- Propagate a WebLogic Server domain to multiple destinations using predefined configuration and extension templates. See "Creating and Configuring WebLogic Domains Using WLST Offline" on page 3-1.
- Retrieve domain configuration and runtime information. See "Navigating and Interrogating MBeans" on page 4-1.
- Edit the domain configuration and persist the changes in the domain's configuration files. See "Editing Configuration MBeans" on page 4-12.
- Edit custom, user-created MBeans and non-WebLogic Server MBeans, such as WebLogic Integration Server and WebLogic Portal Server MBeans. See "Accessing Custom MBeans" on page 4-11.
- Automate domain configuration tasks and application deployment. See "Automating WebLogic Server Administration Tasks" on page 6-1.
- Control and manage the server life cycle. See "Managing Servers and Server Life Cycle" on page 5-1.
- Access the Node Manager and start, stop, and suspend server instances remotely or locally, without requiring the presence of a running Administration Server. See "Using WLST and Node Manager to Manage Servers" on page 5-4.

WLST functionality includes the capabilities of these WebLogic Server command-line utilities: the `weblogic.Admin` utility that you use to interrogate MBeans and configure a WebLogic Server instance (deprecated in this release of WebLogic Server), the `wlconfig` Ant task tool for making WebLogic Server configuration changes, and the `weblogic.Deployer` utility for deploying applications. For more information about these command-line utilities, see:

- "Using Ant Tasks to Configure and Use a WebLogic Server Domain" in *Developing Applications with WebLogic Server*, describes using WebLogic Ant tasks for starting and stopping WebLogic Server instances and configuring WebLogic Server domains.
- "Overview of Deployment Tools" in *Deploying Applications to WebLogic Server*, describes several tools that WebLogic Server provides for deploying applications and stand-alone modules.

You can create, configure, and manage domains using WLST, command-line utilities, and the Administration Console interchangeably. The method that you choose depends on whether you

prefer using a graphical or command-line interface, and whether you can automate your tasks by using a script. See "Script Mode" on page 2-4.

How Does WLST Work?

You can use the scripting tool **online** (connected to a running Administration Server or Managed Server instance) and **offline** (not connected to a running server). For information on WLST online and offline commands, see "WLST Online and Offline Command Summary" on page B-1.

Using WLST Online

Online, WLST provides simplified access to Managed Beans (MBeans), Java objects that provide a management interface for an underlying resource that you can manage through JMX. WLST is a JMX client; all the tasks you can do using WLST online, can also be done programmatically using JMX.

For information on using JMX to manage WebLogic Server resources, see *Developing Custom Management Utilities with JMX*.

When WLST is connected to an Administration Server instance, the scripting tool lets you navigate and interrogate MBeans, and supply configuration data to the server. When WLST is connected to a Managed Server instance, its functionality is limited to browsing the MBean hierarchy.

While you cannot use WLST to change the values of MBeans on Managed Servers, it is possible to use the Management APIs to do so. BEA Systems recommends that you change only the values of configuration MBeans on the Administration Server. Changing the values of MBeans on Managed Servers can lead to an inconsistent domain configuration.

Using WLST Offline

Using WLST offline, you can create a new domain or update an existing domain without connecting to a running WebLogic Server—supporting the same functionality as the Configuration Wizard.

Offline, WLST only provides access to persisted configuration information. You can create new configuration information, and retrieve and change existing configuration information that is persisted in the domain configuration files (located in the `config` directory, for example, `config.xml`) or in a domain template JAR created using Template Builder.

Note: Because WLST offline enables you to access and update the configuration objects that appear in the configuration files only, if you wish to view and/or change attribute values

for a configuration object that is not already persisted in the configuration files as an XML element, you must first create the configuration object.

Modes of Operation

WLST is a command-line interpreter that interprets commands either interactively, supplied one-at-a-time from a command prompt, or in batches, supplied in a file (script), or embedded in your Java code. The modes of operation represent methods for issuing WLST commands:

- Interactively, on the command line—"Interactive Mode"
- In a text file—"Script Mode"
- Embedded in Java code—"Embedded Mode"

Interactive Mode

Interactive mode, in which you enter a command and view the response at a command-line prompt, is useful for learning the tool, prototyping command syntax, and verifying configuration options before building a script. Using WLST interactively is particularly useful for getting immediate feedback after making a critical configuration change. The WLST scripting shell maintains a persistent connection with an instance of WebLogic Server. Because a persistent connection is maintained throughout the user session, you can capture multiple steps that are performed against the server. See "Recording User Interactions" on page 2-16.

In addition, each command that you enter for a WebLogic Server instance uses the same connection that has already been established, eliminating the need for user re-authentication and a separate JVM to execute the command.

Script Mode

Scripts invoke a sequence of WLST commands without requiring your input, much like a shell script. Scripts contain WLST commands in a text file with a `.py` file extension, for example, *`filename`*`.py`. You use script files with the Jython commands for running scripts. See "Running Scripts" on page 2-10.

Using WLST scripts, you can:

- Automate WebLogic Server configuration and application deployment
- Apply the same configuration settings, iteratively, across multiple nodes of a topology

- Take advantage of scripting language features, such as loops, flow control constructs, conditional statements, and variable evaluations that are limited in interactive mode
- Schedule scripts to run at various times
- Automate repetitive tasks and complex procedures
- Configure an application in a hands-free data center

In Listing 2-1, WLST connects to a running Administration Server instance, creates 10 Managed Servers and two clusters, and assigns the servers to a cluster.

Edit the script to contain the username, password, and URL of the Administration Server and start the server before running this script. See "Running Scripts" on page 2-10.

Listing 2-1 Creating a Clustered Domain

```
from java.util import *
from javax.management import *
import javax.management.Attribute

print 'starting the script .... '

connect('username','password','t3://localhost:7001')
clusters = "cluster1","cluster2"
ms1 = {'managed1':7701,'managed2':7702,'managed3':7703, 'managed4':7704,
'managed5':7705}
ms2 = {'managed6':7706,'managed7':7707,'managed8':7708, 'managed9':7709,
'managed10':7710}

clustHM = HashMap()
edit()
startEdit()

for c in clusters:
  print 'creating cluster '+c
  clu = create(c,'Cluster')
  clustHM.put(c,clu)

cd('..\..')

clus1 = clustHM.get(clusters[0])
clus2 = clustHM.get(clusters[1])

for m, lp in ms1.items():
  managedServer = create(m,'Server')
  print 'creating managed server '+m
```

```
  managedServer.setListenPort(lp)
  managedServer.setCluster(clus1)

for m1, lp1 in ms2.items():
  managedServer = create(m1,'Server')
  print 'creating managed server '+m1
  managedServer.setListenPort(lp1)
  managedServer.setCluster(clus2)
save()
activate(block="true")
disconnect()
print 'End of script ...'
exit()
```

Embedded Mode

In embedded mode, you instantiate an instance of the WLST interpreter in your Java code and use it to run WLST commands and scripts. All WLST commands and variables that you use in interactive and script mode can be run in embedded mode.

Listing 2-2 illustrates how to instantiate an instance of the WLST interpreter and use it to connect to a running server, create two servers, and assign them to clusters.

Listing 2-2 Running WLST From a Java Class

```
package wlst;
import java.util.*;
import weblogic.management.scripting.utils.WLSTInterpreter;
import org.python.util.InteractiveInterpreter;

/**
 * Simple embedded WLST example that will connect WLST to a running server,
 * create two servers, and assign them to a newly created cluster and exit.
 * <p>Title: EmbeddedWLST.java</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * <p>Company: BEA Systems</p>
 * @author Satya Ghattu (sghattu@bea.com)
 */

public class EmbeddedWLST
{
  static InteractiveInterpreter interpreter = null;
```

```
  EmbeddedWLST() {
    interpreter = new WLSTInterpreter();
  }

private static void connect() {
    StringBuffer buffer = new StringBuffer();
    buffer.append("connect('weblogic','weblogic')");
    interpreter.exec(buffer.toString());
  }

private static void createServers() {
    StringBuffer buf = new StringBuffer();
    buf.append(startTransaction());
    buf.append("man1=create('msEmbedded1','Server')\n");
    buf.append("man2=create('msEmbedded2','Server')\n");
    buf.append("clus=create('clusterEmbedded','Cluster')\n");
    buf.append("man1.setListenPort(8001)\n");
    buf.append("man2.setListenPort(9001)\n");
    buf.append("man1.setCluster(clus)\n");
    buf.append("man2.setCluster(clus)\n");
    buf.append(endTransaction());
    buf.append("print 'Script ran successfully ...' \n");
    interpreter.exec(buf.toString());
  }

private static String startTransaction() {
    StringBuffer buf = new StringBuffer();
    buf.append("edit()\n");
    buf.append("startEdit()\n");
    return buf.toString();
  }

private static String endTransaction() {
    StringBuffer buf = new StringBuffer();
    buf.append("save()\n");
    buf.append("activate(block='true')\n");
    return buf.toString();
  }

  public static void main(String[] args) {
    new EmbeddedWLST();
    connect();
    createServers();
  }
}
```

Main Steps for Using WLST

The following sections summarize the steps for setting up and using WLST:

Setting Up Your Environment

To set up your environment for WLST:

1. Install and configure the WebLogic Server software, as described in the *WebLogic Server Installation Guide*.
2. Add WebLogic Server classes to the `CLASSPATH` environment variable and *`WL_HOME`*`\server\bin` to the `PATH` environment variable, where *`WL_HOME`* refers to the top-level installation directory for WebLogic Server.

   You can use a *`WL_HOME`*`\server\bin\setWLSEnv` script to set both variables.

   On Windows, a shortcut on the Start menu sets the environment variables and invokes WLST (Tools→WebLogic Scripting Tool).

Invoking WLST

To invoke WLST:

1. If you will be connecting to a WebLogic Server instance through an SSL listen port on a server that is using the demonstration SSL keys and certificates, invoke WLST using the following command:

   ```
   java -Dweblogic.security.SSL.ignoreHostnameVerification=true
   -Dweblogic.security.TrustKeyStore=DemoTrust weblogic.WLST
   ```

   Otherwise, at a command prompt, enter the following command:

```
java weblogic.WLST
```

A welcome message and the WLST prompt appears:

```
wls:/(offline)>
```

2. To use WLST offline, enter commands, set variables, or run a script at the WLST prompt.

   For more information, see "Creating and Configuring WebLogic Domains Using WLST Offline" on page 3-1.

   To use WLST online, start a WebLogic Server instance (see Starting and Stopping Servers) and connect WLST to the server using the `connect` command.

   ```
   wls:/(offline)> connect('username','password','t3s://localhost:7002')
   Connecting to weblogic server instance running at t3s://localhost:7002
   as username weblogic ...
   Successfully connected to Admin Server 'myserver' that belongs to
   domain 'mydomain'.

   wls:/mydomain/serverConfig>
   ```

   Note: BEA Systems strongly recommends that you connect WLST to the server through an SSL port or the administration port. If you do not, the following warning message is displayed:

   ```
   Warning: An insecure protocol was used to connect to the server.
   To ensure on-the-wire security, the SSL port or Admin port should be
   used instead.
   ```

   For detailed information about the `connect` command, see "connect" on page A-10.

Requirements for Entering WLST Commands

Follow these rules when entering WLST commands. Also see "WLST Command and Variable Reference" on page A-1 and "WLST Online and Offline Command Summary" on page B-1.

- Command names and arguments are case sensitive.
- Enclose arguments in single or double quotes. For example, `'newServer'` or `"newServer"`.
- If you specify a backslash character (`\`) in a string, for example when specifying a file pathname, you should precede the quoted string by **r** to instruct WLST to interpret the string in its raw form and ensure that the backslash is taken literally. This format represents standard Jython syntax. For example:

  ```
  readTemplate(r'c:\mytemplate.jar')
  ```

- When using WLST offline, the following characters are not valid in object names: period (`.`), forward slash (`/`), or backward slash (`\`).

  If you need to `cd` to an object name that includes a forward slash (`/`) in its name, include the configuration object name in parentheses. For example:

  ```
  cd('JMSQueue/(jms/REGISTRATION_MDB_QUEUE)')
  ```

- Using WLST and a domain template, you can only create and access security information when you are creating a new domain. When you are updating a domain, you cannot access security information through WLST.
- Display help information for WLST commands by entering the `help` command. See "Getting Help" on page 2-16.

Running Scripts

WLST incorporates two Jython functions that support running scripts:

- `java weblogic.WLST` *`filePath`*`.py`

  This command invokes WLST and executes a script file in a single command. See "Invoke WLST and Run a Script" on page 2-10.

- `execfile(`*`filePath`*`.py)`

  This command executes a script file after you invoke WLST. See "Run a Script From WLST" on page 2-11.

To run the script examples in this guide, copy and save the commands in a text file with a `.py` file extension, for example, *`filename`*`.py`. Use the text file with the commands for running scripts that are listed below. There are sample scripts that you can use as a template when creating a *`script`*`.py` file from scratch. For more information, see "WLST Sample Scripts" on page 1-3.

If the script will connect WLST to a running server instance, start WebLogic Server before running the script.

Invoke WLST and Run a Script

The following command invokes WLST, executes the specified script, and exits the WLST scripting shell. To prevent exiting WLST, use the `-i` flag.

```
java weblogic.WLST filePath.py
java weblogic.WLST -i filePath.py
```

Note: If a WLST script named `wlstProfile.py` exists in the directory from which you invoke WLST or in `user.home` (the home directory of the operating system user account as

determined by the JVM), the commands will be executed automatically once WLST is invoked. In this case, you do not need to specify the name of the WLST script file on the command-line.

For example:

```
c:\>java weblogic.WLST c:/temp/example.py
Initializing WebLogic Scripting Tool (WLST) ...
starting the script ...
...
```

Run a Script From WLST

Use the following command to execute the specified script after invoking WLST.

```
execfile(filePath)
```

For example:

```
c:\>java weblogic.WLST
Initializing WebLogic Scripting Tool (WLST) ...
...
...
wls:/(offline)>execfile('c:/temp/example.py')
starting the script ...
...
```

Importing WLST as a Jython Module

Advanced users can import WLST from WebLogic Server as a Jython module. After importing WLST, you can use it with your other Jython modules and invoke Jython commands directly using Jython syntax.

The main steps include converting WLST definitions and method declarations to a `.py` file, importing the WLST file into your Jython modules, and referencing WLST from the imported file.

To import WLST as a Jython module:

1. Invoke WLST.

   ```
   c:\>java weblogic.WLST
   wls:/(offline)>
   ```

2. Use the `writeIniFile` command to convert WLST definitions and method declarations to a `.py` file.

```
wls:/(offline)> writeIniFile("wl.py")
The Ini file is successfully written to wl.py
wls:/(offline)>
```

3. Open a new command shell and invoke Jython directly by entering the following command:

   ```
   c:\>java org.python.util.jython
   ```

   The Jython package manager processes the JAR files in your classpath. The Jython prompt appears:

   ```
   >>>
   ```

4. Import the WLST module into your Jython module using the Jython `import` command.

   ```
   >>>import wl
   ```

5. Now you can use WLST methods in the module. For example, to connect WLST to a server instance:

   ```
   wl.connect('username','password')
   ....
   ```

   Note: When using WLST as a Jython module, in all WLST commands that have a block argument, block is always set to true, specifying that WLST will block user interaction until the command completes. See "WLST Command and Variable Reference" on page A-1.

Exiting WLST

To exit WLST:

```
wls:/mydomain/serverConfig> exit()
Exiting WebLogic Scripting Tool ...
c:\>
```

Running WLST from Ant

The `wlst` Ant task is predefined in the version of Ant shipped with WebLogic Server and enables you to run WLST scripts from within your Ant build file.

Note: If you want to use the task with your own Ant installation, add the following task definition in your build file:

```
<taskdef name="wlst"
classname="weblogic.ant.taskdefs.management.WLSTTask" />
```

To run WLST from an Ant script:

1. Set your environment in a command shell.
   - On Windows NT, execute the `setWLSEnv.cmd` command, located in the directory *WL_HOME*`\server\bin`, where *WL_HOME* is the top-level directory of your WebLogic Server installation.
   - On UNIX, execute the `setWLSEnv.sh` command, located in the directory *WL_HOME*`/server/bin`, where *WL_HOME* is the top-level directory of your WebLogic Server installation.
2. Add a call to the `wlst` Ant task to execute WLST commands. For example:

```
<target name="runwlst">
   <wlst fileName="./myscript.py" executeScriptBeforeFile="true"
    debug="false" failOnError="false">
     <script>
        connect('weblogic','weblogic','t3://localhost:7001')
     </script>
   </wlst>
</target>
```

   Using the `wlst` Ant task, you can specify WLST commands within a WLST script, using the `fileName` attribute, or embed WLST script commands inside the build file, within the `<script>` tags. For more information about the Ant task command syntax, see "wlst Ant Task Syntax" on page 2-13.
3. Execute the Ant task or tasks specified in the `build.xml` file by typing `ant` in the staging directory, optionally passing the command a target argument:

```
prompt> ant runwlst
```

   Note: Use `ant -verbose` to obtain more detailed messages from the `wlst` task.

wlst Ant Task Syntax

The syntax of the `wlst` Ant task is as follows:

```
<wlst propertiesFile="propsFile" fileName="fileName" arguments="arglist"
failOnError="value" executeScriptBeforeFile="value" debug="value">
<script>
   wlst-commands
</script>
```

You can specify WLST commands within a WLST script (`.py`) file and specify that file using `fileName` attribute, or embed WLST script commands inside the build file, enclosing them within the `<script>` tags.

The following table defines the Ant task attributes that you can specify as part of the `wlst` Ant task.

Table 2-1 Ant Task Attributes

Attribute	Definition
`propertiesFile="`*`propsFile`*`"`	Optional. Name and location of the properties file that contains name-value pairs that you can reference in your script.
`fileName="`*`fileName`*`"`	Optional. Name and location of the WLST script file that you would like to execute. If the specified WLST script file does not exist, the `wlst` Ant task fails.
`arguments="`*`arglist`*`"`	Optional. List of arguments to pass to the script. These arguments are accessible using the `sys.argv` variable.
`failOnError="`*`value`*`"`	Optional. Boolean value specifying whether the Ant build script will fail if the `wlst` Ant task fails. This attribute defaults to `true`.
`executeScriptBeforeFile="`*`value`*`"`	Optional. Boolean value specifying whether the embedded script is executed before the specified WLST script file. This attribute defaults to `true`, specifying that the embedded script is executed first.
`debug="`*`value`*`"`	Optional. Boolean value specifying whether the embedded script is executed before the specified WLST script file. This attribute defaults to `false`.

Examples

Example 1

In the following example, the `createServer` target:

- Executes the embedded script to connect to the server at `t3://localhost():7001`, edits the configuration to create a new server, and saves and activates the configuration changes. (Note that `executeScriptBeforeFile` is set to `true`, so the embedded script is executed first, before the specified WLST script file.)
- Executes the WLST script file `myscript.py` that is located in the currrent directory.
- Defines three arguments that are passed to the script. These arguments are accessible using the `sys.argv` variable.
- Continues execution, as per the `failOnError="`false`"` setting, even if the `wlst` Ant task fails to execute.

- Disables debugging.

```
<target name="configServer">
   <wlst debug="false" failOnError="false" executeScriptBeforeFile="true"
    fileName="./myscript.py">
      <script>
         connect('weblogic','weblogic','t3://localhost:7001')
     </script>
   </wlst>
</target>
```

Example 2

In the following example, the `loop` target:

- Executes the WLST script file `myscript.py` in the currrent directory. (Note that `executeScriptBeforeFile` is set to `false`, so the WLST script file is executed first, before the embedded script.)
- Executes the embedded script to connect to the server at `t3://localhost():7001` and access and print the list of servers in the domain.
- Results in a build failure if the `wlst` task fails to execute, as per the `failOnError="true"` setting.
- Enables debugging.

```
<target name="loop">
   <wlst debug="true" executeScriptBeforeFile="false"
    fileName="./myscript.py" failOnError="true">
      <script>
         print 'In the target loop'
         connect('weblogic','weblogic','t3://localhost:7001')
         svrs = cmo.getServers()
         print 'Servers in the domain are'
         for x in svrs: print x.getName()
      </script>
   </wlst>
</target>
```

Example 3

In the following example, the `error` target:

- Executes the embedded script to print the variable, `thisWillCauseNameError`.
- Continues execution, as per the `failOnError="false"` setting, even if the `thisWillCauseNameError` variable does not exist and the `wlst` Ant task fails to execute.
- Enables debugging.

```
<target name="error">
   <wlst debug="true" failOnError="false">
      <script>print thisWillCauseNameError</script>
   </wlst>
</target>
```

Getting Help

To display information about WLST commands and variables, enter the `help` command.

If you specify the `help` command without arguments, WLST summarizes the command categories. To display information about a particular command, variable, or command category, specify its name as an argument to the `help` command. To list a summary of all online or offline commands from the command line using the following commands, respectively:

```
help('online')
help('offline')
```

The `help` command will support a query; for example, `help('get*')` displays the syntax and usage information for all commands that begin with `get`.

For example, to display information about the `disconnect` command, enter the following command:

```
wls:/mydomain/serverConfig> help('disconnect')
```

The command returns the following:

```
Description:
Disconnect from a weblogic server instance.

Syntax:
disconnect()

Example:
wls:/mydomain/serverConfig> disconnect()
```

Recording User Interactions

To start and stop the recording of all WLST command input, enter:

```
startRecording(recordFilePath,[recordAll])
stopRecording()
```

You must specify the file pathname for storing WLST commands when you enter the `startRecording` command. You can also optionally specify whether or not you want to capture all user interactions, or just the WLST commands; the `recordAll` argument defaults to false.

For example, to record WLST commands in the `record.py` file, enter the following command:

```
wls:/mydomain/serverConfig> startRecording('c:/myScripts/record.py')
```

For more information, see "startRecording" on page A-87 and "stopRecording" on page A-89.

Redirecting WLST Output to a File

To start and stop redirecting WLST output to a file, enter:

```
redirect(outputFile,[toStdOut])
stopRedirect()
```

You must specify the pathname of the file to which you want to redirect WLST output. You can also optionally specify whether you want WLST output to be sent to `stdout`; the `toStdOut` argument defaults to true.

For example, to redirect WLST output to the `logs/wlst.log` file in the current directory and disable output from being sent to `stdout`, enter the following command:

```
wls:/mydomain/serverConfig> redirect('./logs/wlst.log', 'false')
```

For more information, see "redirect" on page A-85 and "stopRedirect" on page A-90.

Converting an Existing Configuration into a WLST Script

To convert an existing server configuration (`config` directory) to an executable WLST script, enter:

```
configToScript([domainDir], [scriptPath], [overwrite], [propertiesFile],
[deploymentScript])
```

You can optionally specify:

- The domain directory that contains the configuration that you want to convert (defaults to `./config`)
- The path to the directory in which you want to write the converted WLST script (defaults to `./config/config.py`)

- Whether to overwrite the script if it already exists (defaults to `true`)
- The path to the directory in which you want to store the properties file (defaults to pathname specified for *`scriptPath`*)
- Whether to create a script that performs deployments only (defaults to `false`)

Note: `configToScript()` creates ancillary user-config and user-key files to hold encrypted attributes.

You can use the resulting script to re-create the resources on other servers. Before running the generated script, you should update the properties file to specify values that are appropriate for your environments. When you run the generated script:

- If a server is currently running, WLST will try to connect using the values in the properties file and then run the script commands to create the server resources.
- If no server is currently running, WLST will start a server with the values in the properties file, run the script commands to create the server resources, and shutdown the server. This may cause WLST to exit from the command shell.

For example, the following command creates a WLST script from the domain located at `c:/bea/user_projects/domains/mydomain`, and saves it to `c:/bea/myscripts`.

```
wls:/(offline)> configToScript('c:/bea/user_projects/domains/mydomain',
'c:/bea/myscripts')
```

For more information, see "configToScript" on page A-72.

Customizing WLST

You can customize WLST using the WLST home directory, which is located at *`WL_HOME`*`/common/wlst`, by default, where *`WL_HOME`* refers to the top-level installation directory for WebLogic Server. All Python scripts that are defined within the WLST home directory are imported at WLST startup.

Note: You can customize the default WLST home directory by passing the following argument on the command line:
`-Dweblogic.wlstHome=<`*`another-directory`*`>`

The following table describes ways to customize WLST.

Table 2-2 Customizing WLST

To define custom...	Do the following...	For a sample script, see...
WLST commands	Create a Python script defining the new commands and copy that file to *WL_HOME*/common/wlst.	*WL_HOME*/common/wlst/sample.py Within this script, the wlstHomeSample() command is defined, which prints a String, as follows: wls:/(offline)> **wlstHomeSample()** Sample wlst home command
WLST commands within a library	Create a Python script defining the new commands and copy that file to *WL_HOME*/common/wlst/lib. The scripts located within this directory are imported as Jython libraries.	*WL_HOME*/common/wlst/lib/wlstLibSample.py Within this script, the wlstExampleCmd() command is defined, which prints a String, as follows: wls:/(offline)> **wlstLibSample.wlstExampleCmd()** Example command
WLST commands as a Jython module	Create a Python script defining the new commands and copy that file to *WL_HOME*/common/wlst/modules. This script can be imported into other Jython modules, as described in "Importing WLST as a Jython Module" on page 2-11.	*WL_HOME*/common/wlst/modules/wlstModule.py A JAR file, jython.jar, containing all of the Jython modules that are available in Jython 2.1 is also available within this directory.

CHAPTER 3

Creating and Configuring WebLogic Domains Using WLST Offline

WLST enables you to create a new domain or update an existing domain without connecting to a running WebLogic Server (that is, using WLST offline)—supporting the same functionality as the Configuration Wizard.

You can create new configuration information, and retrieve and change existing configuration information that is persisted in the domain configuration files (located in the `config` directory, for example, `config.xml`) or in a domain template JAR created using Template Builder.

The following sections describe how to create and configure WebLogic domains using WLST offline. Topics include:

For more information about the Configuration Wizard, see *Creating WebLogic Domains Using the Configuration Wizard*.

Notes: Before creating or updating a domain, you must set up your environment and invoke WLST, as described in "Main Steps for Using WLST" on page 2-8. You can exit WLST at anytime using the `exit` command, as follows: `exit()`

When using the Configuration Wizard or WLST Offline to create or extend a clustered domain with a template that has applications containing application-scoped JDBC and/or JMS resources, you may need to perform additional steps (after the domain is created or extended) to make sure that the application and its application-scoped resources are targeted and deployed properly in a clustered environment. For more information on the targeting and deployment of application-scoped modules, see "Deploying Applications and Modules" in *Deploying Applications to WebLogic Server*.

Creating a Domain (Offline)

To create a domain using WLST offline, use one of the options described in the following table. The first option enables you to create a domain from a template, quickly and easily, but does not enable you modify the configuration settings defined in the template at the time of creation. The second option enables you to modify the configuration settings when creating the domain.

Table 3-1 Creating a Domain (Offline)

Option	To...	Use the following command(s)...	For more information, see...
1	Create a new domain using a specified template.	`createDomain(domainTemplate, domainDir, user, password)`	"createDomain" on page A-14

Table 3-1 Creating a Domain (Offline) (Continued)

Option	To...	Use the following command(s)...	For more information, see...
2	1. Open an existing domain template for domain creation	`readTemplate(templateFileName)`	"readTemplate" on page A-18
	2. Modify the domain (optional)	Various commands **Note:** Because WLST offline enables you to access and update the configuration objects that appear in the configuration files only, if you wish to view and/or change attribute values for a configuration object that is not already persisted in the configuration files as an XML element, you must first create the configuration object. For more information, see "create" on page A-47.	"Browsing and Accessing Information About the Configuration Bean Hierarchy (Offline)" on page 3-5 "Editing a Domain (Offline)" on page 3-5.
	3. Set the password for the default user, if it is not already set **Note:** The default username and password must be set before you can write the domain.	`cd('/Security/domainname/User/username')` `cmo.setPassword('password')`	"Stepping Through a Sample Script: Creating a Domain Using WLST Offline" on page 3-8
	4. Write the domain configuration information to the specified directory	`writeDomain(domainDir)`	"writeDomain" on page A-19
	5. Close the current domain template	`closeTemplate()`	"closeTemplate" on page A-10

Updating an Existing Domain (Offline)

To update an existing domain using WLST offline, you perform the steps defined in the following table.

Table 3-2 Steps for Updating an Existing Domain (Offline)

Step	To...	Use this command...	For more information, see...
1	Open an existing domain for update	`readDomain(domainDirName)`	"readDomain" on page A-17
2	Extend the current domain (optional)	`addTemplate(templateFileName)`	"addTemplate" on page A-8
3	Modify the domain (optional)	Various commands **Note:** Because WLST offline enables you to access and update the configuration objects that appear in the configuration files only, if you wish to view and/or change attribute values for a configuration object that is not already persisted in the configuration files as an XML element, you must first create the configuration object. For more information, see "create" on page A-47.	"Browsing and Accessing Information About the Configuration Bean Hierarchy (Offline)" on page 3-5 "Editing a Domain (Offline)" on page 3-5.
4	Save the domain	`updateDomain()`	"updateDomain" on page A-19
5	Close the domain	`closeDomain()`	"closeDomain" on page A-9

Browsing and Accessing Information About the Configuration Bean Hierarchy (Offline)

To browse and access information about the configuration bean hierarchy using WLST offline, you can perform any of the tasks defined in the following table.

Note: Because WLST offline enables you to access and update the configuration objects that appear in the configuration files only, if you wish to view and/or change attribute values for a configuration object that is not already persisted in the configuration files as an XML element, you must first create the configuration object.

Table 3-3 Displaying Domain Configuration Information (Offline)

To...	Use this command...	For more information, see...
Navigate the hierarchy of configuration beans	`cd(path)`	"cd" on page A-3
List child attributes or configuration beans for the current configuration bean	`ls(['a' | 'c'])`	"ls" on page A-81
Toggle the display of the configuration bean navigation path information at the prompt	`prompt(['off'|'on'])`	"prompt" on page A-5
Display the current location in the configuration bean hierarchy	`pwd()`	"pwd" on page A-6
Display all variables used by WLST	`dumpVariables()`	"dumpVariables" on page A-74
Display the stack trace from the last exception that occurred while performing a WLST action	`dumpStack()`	"dumpStack" on page A-74

Editing a Domain (Offline)

To edit a domain using WLST offline, you can perform any of the tasks defined in the following table.

Note: Because WLST offline enables you to access and update the configuration objects that appear in the configuration files only, if you wish to view and/or change attribute values for a configuration object that is not already persisted in the configuration files as an XML element, you must first create the configuration object.

For more information about:

- Creating MBeans, see "Understanding WebLogic Server MBeans" in *Developing Custom Management Utilities with JMX*.
- Examples of creating specific types of MBean resources, for example, a JMS or JDBC system resource, refer to the WLST sample scripts installed with your product, as described in "WLST Sample Scripts" on page 1-3.
- MBeans, their child types, attributes, and operations, see *WebLogic Server MBean Reference*.

Table 3-4 Editing a Domain

To...	Use this command...	For more information, see...
Assign configuration beans	`assign(sourceType, sourceName, destinationType, destinationName)`	"assign" on page A-42
Unassign configuration beans	`unassign(sourceType, sourceName, destinationType, destinationName)`	"unassign" on page A-64
Create and delete configuration beans	`create(name, childMBeanType)` `delete(name, childMBeanType)`	"create" on page A-47 "delete" on page A-49
Get and set attribute values	`get(attrName)` `set(attrName, value)`	"get" on page A-51 "set" on page A-57
Set configuration options	`setOption(optionName, value)`	"setOption" on page A-58
Load SQL files into a database	`loadDB(dbVersion, connectionPoolName)`	"loadDB" on page A-55

Creating a Domain Template (Offline)

To create a domain template using WLST offline, perform the steps described in the following table.

Note: You can also use the `pack` command to create a domain template quickly and easily. For more information, see *Creating Templates and Domains Using the pack and unpack Commands*.

Table 3-5 Steps for Creating a Domain Template (Offline)

Step	To...	Use this command...	For more information, see...
1	Open an existing domain or template	`readDomain(domainDirName)` `readTemplate(templateFileName)`	"readDomain" on page A-17 "readTemplate" on page A-18
2	Set the password for the default user, if it is not already set **Note:** The default username and password must be set before you can write the domain template.	`cd('/Security/domainname/User/username')` `cmo.setPassword('password')`	"Stepping Through a Sample Script: Creating a Domain Using WLST Offline" on page 3-8
3	Write the domain configuration information to a domain template	`writeTemplate(templateName)`	"writeTemplate" on page A-21

Exporting Diagnostic Data (Offline)

To export diagnostic information using WLST offline, use the `exportDiagnosticData` command defined in the following table.

The results are saved to an XML file. For more information about the WebLogic Server Diagnostic Service, see *Configuring and Using the WebLogic Diagnostic Service*.

Table 3-6 Exporting Diagnostic Data

To...	Use this command...	For more information, see...
Execute a query against the specified log file	`exportDiagnosticData([options])`	"exportDiagnosticData" on page A-36

Stepping Through a Sample Script: Creating a Domain Using WLST Offline

The following example steps through a sample script that creates a WebLogic domain using the Basic WebLogic Server Domain template. The sample script demonstrates how to open a domain template; create and edit configuration objects; write the domain configuration information to the specified directory; and close the domain template.

The script shown is installed with your product at *WL_HOME*\common\templates\scripts\wlst\basicWLSDomain.py, where *WL_HOME* refers to the top-level installation directory for WebLogic Server. See also "WLST Offline Sample Scripts" on page 1-4 for a description of the full set of sample scripts that are installed with your product.

To create a WebLogic domain using the Basic WebLogic Server Domain template:

1. Open an existing domain template (assuming WebLogic Server is installed at c:/bea/weblogic91). In this example, we open the Basic WebLogic Server Domain template.

   ```
   readTemplate('c:/bea/weblogic91/common/templates/domains/wls.jar')
   ```

   Note: When you open a template or domain, WLST is placed into the configuration bean hierarchy for that domain, and the prompt is updated to reflect the current location in the configuration hierarchy. For example:

   ```
   wls:/offline/base_domain>
   ```

   WebLogic Server configuration beans exist within a hierarchical structure. Similar to a file system, the hierarchies correspond to drives; types and instances are directories; attributes and operations are files. WLST traverses the hierarchical structure of configuration beans using commands such as cd, ls, and pwd in a similar way that you would navigate a file system in a UNIX or Windows command shell. After navigating to an configuration bean instance, you interact with the bean using WLST commands. For more information, see "Navigating and Interrogating MBeans" on page 4-1.

2. Configure the domain.

   a. Configure the Administration Server and SSL port.

   ```
   cd('Servers/AdminServer')
   set('ListenAddress','')
   set('ListenPort', 7001)
   ```

```
create('AdminServer','SSL')
cd('SSL/AdminServer')
set('Enabled', 'True')
set('ListenPort', 7002)
```

b. Define the default user password.

```
cd('/')
cd('Security/base_domain/User/weblogic')
cmo.setPassword('weblogic')
```

c. Create a JMS Server.

```
cd('/')
create('myJMSServer', 'JMSServer')
```

d. Create a JMS system resource.

```
cd('/')
create('myJmsSystemResource', 'JMSSystemResource')
cd('JMSSystemResource/myJmsSystemResource/JmsResource/NO_NAME_0')
```

e. Create a JMS queue and its subdeployment.

```
myq=create('myQueue','Queue')
myq.setJNDIName('jms/myqueue')
myq.setSubDeploymentName('myQueueSubDeployment')
cd('/')
cd('JMSSystemResource/myJmsSystemResource')
create('myQueueSubDeployment', 'SubDeployment')
```

f. Create a JDBC data source, configure the JDBC driver, and create a new JDBC user.

```
cd('/')
create('myDataSource', 'JDBCSystemResource')
cd('JDBCSystemResource/myDataSource/JdbcResource/myDataSource')
create('myJdbcDriverParams','JDBCDriverParams')
cd('JDBCDriverParams/NO_NAME_0')
set('DriverName','com.pointbase.jdbc.jdbcUniversalDriver')
set('URL','jdbc:pointbase:server://localhost/demo')

set('PasswordEncrypted', 'PBPUBLIC')
set('UseXADataSourceInterface', 'false')
create('myProps','Properties')
cd('Properties/NO_NAME_0')
create('user', 'Property')
cd('Property/user')
cmo.setValue('PBPUBLIC')

cd('/JDBCSystemResource/myDataSource/JdbcResource/myDataSource')
create('myJdbcDataSourceParams','JDBCDataSourceParams')
```

```
cd('JDBCDataSourceParams/NO_NAME_0')
set('JNDIName', java.lang.String("myDataSource_jndi"))

cd('/JDBCSystemResource/myDataSource/JdbcResource/myDataSource')
create('myJdbcConnectionPoolParams','JDBCConnectionPoolParams')
cd('JDBCConnectionPoolParams/NO_NAME_0')
set('TestTableName','SYSTABLES')
```

g. Target the resources.

```
cd('/')
assign('JMSServer', 'myJMSServer', 'Target', 'AdminServer')
assign('JMSSystemResource.SubDeployment',
'myJmsSystemResource.myQueueSubDeployment', 'Target', 'myJMSServer')
assign('JDBCSystemResource', 'myDataSource', 'Target', 'AdminServer')
```

3. Save the domain.

```
setOption('OverwriteDomain', 'true')
writeDomain('c:/bea/user_projects/domains/wls_testscript')
```

4. Close the current domain template.

```
closeTemplate()
```

5. Exit WLST.

```
exit()
```

CHAPTER 4

Navigating and Editing MBeans

The following sections describe how to navigate, interrogate, and edit MBeans using WLST:

Navigating and Interrogating MBeans

WLST provides simplified access to MBeans. While the `weblogic.Admin` utility (deprecated in this release of WebLogic Server) and WLST provide an interface for interacting with MBeans, the manner in which you retrieve MBeans is different.

The MBean-related commands that the `weblogic.Admin` utility provides require you to determine the object name of the MBean with which you want to interact. While the object name is generated based on documented conventions, it can be difficult to accurately determine the object names of child MBeans several layers within a hierarchy.

WLST offers a different style of retrieving MBeans: instead of providing object names, you navigate a hierarchy of MBeans in a similar fashion to navigating a hierarchy of files in a file system.

WebLogic Server organizes its MBeans in a hierarchical data model. In the WLST file system, MBean hierarchies correspond to drives; MBean types and instances are directories; MBean attributes and operations are files. WLST traverses the hierarchical structure of MBeans using commands such as `cd`, `ls`, and `pwd` in a similar way that you would navigate a file system in a UNIX or Windows command shell. After navigating to an MBean instance, you interact with the MBean using WLST commands.

Note: Because WLST offline enables you to access and update the configuration objects that appear in the configuration files only, if you wish to view and/or change attribute values for a configuration object that is not already persisted in the configuration files as an XML element, you must first create the configuration object.

In the configuration hierarchy, the root directory is `DomainMBean` (see DomainMBean in the *WebLogic Server MBean Reference*); the MBean type is a subdirectory under the root directory; the name of the MBean (the MBean instance) is a subdirectory under the MBean type directory; and MBean attributes and operations are nodes (like files) under the MBean directory. Each MBean instance is a subdirectory of an MBean type. In most cases, there can be multiple instances of a type.

Figure 4-1 Configuration MBean Hierarchy

```
Domain MBean (root)
    |- - - MBean type (LogMBean)
                |- - - MBean instance (medrec)
                            |- - - MBean attributes & operations (FileName)
    |- - - MBean type (SecurityConfigurationMBean)
    |- - - MBean type (ServerMBean)
                |- - - MBean instance (MedRecServer)
                            |- - - MBean attributes & operations (StartupMode)
                |- - - MBean instance (ManagedServer1)
                            |- - - MBean attributes & operations (AutoRestart)
```

WLST first connects to a WebLogic Server instance at the root of the server's configuration MBeans, a single hierarchy whose root is `DomainMBean`. WLST commands provide access to all the WebLogic Server MBean hierarchies within a domain, such as a server's runtime MBeans, runtime MBeans for domain-wide services, and an editable copy of all the configuration MBeans in the domain. For more information, see "Tree Commands" on page A-117.

For more information about MBean hierarchies, see "WebLogic Server MBean Data Model" in *Developing Custom Management Utilities with JMX*.

Changing the Current Management Object

When WLST first connects to an instance of WebLogic Server, the variable, `cmo` (Current Management Object), is initialized to the root of all configuration management objects, `DomainMBean`. When you navigate to an MBean type, the value of `cmo` reflects the parent MBean. When you navigate to an MBean instance, WLST changes the value of `cmo` to be the current MBean instance, as shown in Listing 4-1.

For more information on WLST variables, see "WLST Variable Reference" on page A-127.

Listing 4-1 Changing the Current Management Object

```
C:\> java weblogic.WLST
Initializing WebLogic Scripting Tool (WLST) ...
Welcome to Weblogic Server Administration Scripting Shell
...
wls:/(offline)> connect('username','password')
Connecting to weblogic server instance running at t3://localhost:7001 as
username weblogic ...
Successfully connected to Admin Server 'myserver' that belongs to domain
'mydomain'.
Warning: An insecure protocol was used to connect to the server.
To ensure on-the-wire security, the SSL port or Admin port should be used
instead.
wls:/mydomain/serverConfig> cmo
[MBeanServerInvocationHandler]mydomain:Name=mydomain,Type=Domain
wls:/mydomain/serverConfig> cd('Servers')
wls:/mydomain/serverConfig/Servers> cmo
[MBeanServerInvocationHandler]mydomain:Name=mydomain,Type=Domain
wls:/mydomain/serverConfig/Servers> cd('myserver')
wls:/mydomain/serverConfig/Servers/myserver> cmo
[MBeanServerInvocationHandler]mydomain:Name=myserver,Type=Server
```

After navigating to an instance of `ServerMBean`, WLST changes the value of `cmo` from `DomainMBean` to `ServerMBean`.

Navigating and Displaying Configuration MBeans Example

The commands in Listing 4-2 instruct WLST to connect to an Administration Server instance, navigate, and display MBeans in `DomainMBean`. If no argument is specified, the `ls` command lists all the child MBeans and attributes.

Listing 4-2 Navigating and Displaying Configuration MBeans

```
C:\> java weblogic.WLST
wls:/offline> connect('username','password')
wls:/mydomain/serverConfig> ls()
dr--   AppDeployments
dr--   BridgeDestinations
dr--   Clusters
dr--   DeploymentConfiguration
dr--   Deployments
dr--   EmbeddedLDAP
...
-r--   AdminServerName                           myserver
-r--   AdministrationMBeanAuditingEnabled        false
-r--   AdministrationPort                        9002
-r--   AdministrationPortEnabled                 false
-r--   AdministrationProtocol                    t3s
-r--   ArchiveConfigurationCount                 5
...
wls:/mydomain/serverConfig> cd('Servers')
wls:/mydomain/serverConfig/Servers> ls()
dr--   managed1
dr--   myserver
wls:/mydomain/serverConfig/Servers> cd('myserver')
wls:/mydomain/serverConfig/Servers/myserver> ls()
dr--   COM
dr--   CandidateMachines
dr--   Cluster
dr--   DefaultFileStore
dr--   ExecutiveQueues
dr--   IIOP
dr--   JTAMigrateableTarget
dr--   Log
dr--   Machine
dr--   NetworkAccessPoints
dr--   OverloadProtection
dr--   SSL
...
-r--   AcceptBacklog                             50
-r--   AdminReconnectIntervalSeconds             10
```

```
-r--   AdministrationPort                          0
-r--   AdministrationPortAfterOverride             9002
-r--   AdministrationPortEnabled                   false
-r--   AdministrationProtocol                      t3s
-r--   AutoKillIfFailed                            false
-r--   AutoRestart                                 true
....
wls:/mydomain/serverConfig/Servers/myserver> cd('Log/myserver')
wls:/mydomain/serverConfig/Servers/myserver/Log/myserver> ls()
dr--   DomainLogBroadcastFilter
dr--   LogFileFilter
dr--   MemoryBufferFilter
dr--   StdoutFilter

-r--   DomainLogBroadcastFilter                    null
-r--   DomainLogBroadcastSeverity                  Warning
-r--   DomainLogBroadcasterBufferSize              0
-r--   FileCount                                   7
-r--   FileMinSize                                 500
-r--   FileName                                    myserver.log
-r--   FileTimeSpan                                24
-r--   Log4jLoggingEnabled                         false
-r--   LogFileFilter                               null
-r--   LogFileRotationDir                          null
-r--   LogFileSeverity                             Debug
-r--   MemoryBufferFilter                          null
-r--   MemoryBufferSeverity                        Debug
-r--   MemoryBufferSize                            500
-r--   Name                                        myserver
-r--   Notes                                       null
-r--   NumberOfFilesLimited                        false
-r--   RedirectStderrToServerLogEnabled            false
-r--   RedirectStdoutToServerLogEnabled            false
-r--   RotateLogOnStartup                          true
-r--   RotationTime                                00:00
-r--   RotationType                                bySize
-r--   StdoutFilter                                null
-r--   StdoutSeverity                              Warning
-r--   Type                                        Log

-r-x   isSet                                  Boolean : String(propertyName)
-r-x   unSet                                     Void : String(propertyName)
```

In the `ls` command output information, `d` designates an MBean with which you can use the `cd` command (analogous to a directory in a file system), `r` indicates a readable property, `w` indicates a writeable property, and `x` an executable operation.

Note: The `ls` command property information is based on `MBeanInfo`; it does not reflect user permissions.

To navigate back to a parent MBean, enter the `cd('..')` command:

```
wls:/mydomain/serverConfig/Servers/myserver/Log/myserver> cmo
[MBeanServerInvocationHandler]mydomain:Name=myserver,Server=myserver,Type=
Log
wls:/mydomain/serverConfig/Servers/myserver/Log/myserver> cd('..')
wls:/mydomain/serverConfig/Servers/myserver/Log>
wls:/mydomain/serverConfig/Servers/myserver/Log> cmo
[MBeanServerInvocationHandler]mydomain:Name=myserver,Type=Server
```

After navigating back to the parent MBean type, WLST changes the `cmo` from `LogMBean` to `ServerMBean`.

To get back to the root MBean after navigating to an MBean that is deep in the hierarchy, enter the `cd('/')` command.

Browsing Runtime MBeans

Similar to the configuration information, WebLogic Server runtime MBeans are arranged in a hierarchical data structure. When connected to an Administration Server, you access the runtime MBean hierarchy by entering the `serverRuntime` or the `domainRuntime` command. The `serverRuntime` command places WLST at the root of the server runtime management objects, `ServerRuntimeMBean`; the `domainRuntime` command, at the root of the domain-wide runtime management objects, `DomainRuntimeMBean`. When connected to a Managed Server, the root of the runtime MBeans is `ServerRuntimeMBean`. The domain runtime MBean hierarchy exists on the Administration Server only; you cannot use the `domainRuntime` command when connected to a Managed Server.

For more information, see ServerRuntimeMBean and DomainRuntimeMBean in the *WebLogic Server MBean Reference*.

Using the `cd` command, WLST can navigate to any of the runtime child MBeans. The navigation model for runtime MBeans is the same as the navigation model for configuration MBeans. However, runtime MBeans exist only on the same server instance as their underlying managed resources (except for the domain-wide runtime MBeans on the Administration Server) and they are all un-editable.

Navigating and Displaying Runtime MBeans Example

The commands in Listing 4-3 instruct WLST to connect to an Administration Server instance, navigate, and display server and domain runtime MBeans.

Listing 4-3 Navigating and Displaying Runtime MBeans

```
wls:/(offline) > connect('username','password')
wls:/mydomain/serverConfig> serverRuntime()
Location changed to serverRuntime tree. This is a read-only tree with
ServerRuntimeMBean as the root.
For more help, use help('serverRuntime')
wls:/mydomain/serverRuntime> ls()
dr--   ApplicationRuntimes
dr--   ClusterRuntime
dr--   ConnectorServiceRuntime
...
dr--   JDBCServiceRuntime
dr--   JMSRuntime
dr--   JTARuntime
dr--   JVMRuntime
dr--   LibraryRuntimes
dr--   MailSessionRuntimes
dr--   RequestClassRuntimes
dr--   ServerChannelRuntimes
dr--   ServerSecurityRuntime
dr--   ServerServices
dr--   ThreadPoolRuntime
dr--   WLDFAccessRuntime
dr--   WLDFRuntime
dr--   WTCRuntime
dr--   WorkManagerRuntimes

-r--   ActivationTime                          1093958848908
-r--   AdminServer                             true
-r--   AdminServerHost
-r--   AdminServerListenPort                   7001
-r--   AdminServerListenPortSecure             false
-r--   AdministrationPort                      9002
-r--   AdministrationPortEnabled               false
...
wls:/mydomain/serverRuntime> domainRuntime()
Location changed to domainRuntime tree. This is a read-only tree with
DomainRuntimeMBean as the root.
For more help, use help('domainRuntime')
wls:/mydomain/domainRuntime> ls()
```

```
dr--   DeployerRuntime
...
dr--   ServerLifecycleRuntimes
dr--   ServerRuntimes

-r--   ActivationTime                                Tue Aug 31 09:27:22 EDT 2004

-r--   Clusters                                       null
-rw-   CurrentClusterDeploymentTarget                 null
-rw-   CurrentClusterDeploymentTimeout                0
-rw-   Name                                           mydomain
-rw-   Parent                                         null
-r--   Type                                           DomainRuntime

-r-x   lookupServerLifecycleRuntime                   javax.management.ObjectName

: java.lang.String
wls:/mydomain/domainRuntime>
```

The commands in Listing 4-4 instruct WLST to navigate and display runtime MBeans on a Managed Server instance.

Listing 4-4 Navigating and Displaying Runtime MBeans on a Managed Server

```
wls:/offline> connect('username','password',‘t3://localhost:7701’)
Connecting to weblogic server instance running at t3://localhost:7701 as
username weblogic ...
Successfully connected to managed Server 'managed1' that belongs to domain
'mydomain'.
Warning: An insecure protocol was used to connect to the server.
To ensure on-the-wire security, the SSL port or Admin port should be used
instead.
wls:/mydomain/serverConfig> serverRuntime()
wls:/mydomain/serverRuntime> ls()
dr--   ApplicationRuntimes
dr--   ClusterRuntime
...
dr--   JMSRuntime
dr--   JTARuntime
dr--   JVMRuntime
dr--   LibraryRuntimes
dr--   MailSessionRuntimes
dr--   RequestClassRuntimes
dr--   ServerChannelRuntimes
dr--   ServerSecurityRuntime
```

```
dr--   ThreadPoolRuntime
dr--   WLDFAccessRuntime
dr--   WLDFRuntime
dr--   WTCRuntime
dr--   WorkManagerRuntimes

-r--   ActivationTime                                1093980388931
-r--   AdminServer                                   false
-r--   AdminServerHost                               localhost
-r--   AdminServerListenPort                         7001
-r--   AdminServerListenPortSecure                   false
-r--   AdministrationPort                            9002
-r--   AdministrationPortEnabled                     false
...
wls:/mydomain/serverRuntime>
```

Navigating Among MBean Hierarchies

To navigate to a configuration MBean from the runtime hierarchy, enter the `serverConfig` or `domainConfig` (if connected to an Administration Server only) command. This places WLST at the configuration MBean to which you last navigated before entering the `serverRuntime` or `domainRuntime` command.

The commands in the following example instruct WLST to navigate from the runtime MBean hierarchy to the configuration MBean hierarchy and back:

```
wls:/mydomain/serverRuntime/JVMRuntime/managed1> serverConfig()
Location changed to serverConfig tree. This is a read-only tree with
DomainMBean as the root.
For more help, use help('serverConfig')
wls:/mydomain/serverConfig> cd ('Servers/managed1')
wls:/mydomain/serverConfig/Servers/managed1> cd('Log/managed1')
wls:/mydomain/serverConfig/Servers/managed1/Log/managed1> serverRuntime()
wls:/mydomain/serverRuntime/JVMRuntime/managed1>
```

Entering the `serverConfig` command from the runtime MBean hierarchy again places WLST at the configuration MBean to which you last navigated.

```
wls:/mydomain/serverRuntime/JVMRuntime/managed1> serverConfig()
wls:/mydomain/serverConfig/Servers/managed1/Log/managed1>
```

For more information, see "Tree Commands" on page A-117.

Alternatively, you can use the `currentTree` command to store your current MBean hierarchy location and to return to that location after navigating away from it. See "currentTree" on page A-4.

For example:

```
wls:/mydomain/serverConfig/Servers/managed1/Log/managed1> myLocation =
currentTree()
wls:/mydomain/serverConfig/Servers/managed1/Log/managed1> serverRuntime()
wls:/mydomain/serverRuntime> cd('JVMRuntime/managed1')
wls:/mydomain/serverRuntime/JVMRuntime/managed1>myLocation()
wls:/mydomain/serverConfig/Servers/managed1/Log/managed1>
```

Finding MBeans

To locate a particular MBean and attribute, you use the `find` command. WLST returns the pathname to the MBean that stores the attribute and its value. You can use the `getMBean` command to return the MBean specified by the path. For more information, see "find" on page A-75 and "getMBean" on page A-77.

For example:

```
wls:/mydomain/edit !> find('logfilename')
searching ...
/ApplicationRuntimes/myserver_wlnav.war/WebAppComponentRuntime/myserver_my
server_wlnav.war_wlnav_/wlnavLogFilename                    null
/Servers/myserver                          JDBCLogFileName jdbc.log
/Servers/myserver/WebServer/myserver       LogFileName access.log
wls:/mydomain/edit !> bean=getMBean('Servers/myserver/WebServer/myserver')
wls:/mydomain/edit !> print bean
[MBeanServerInvocationHandler]mydomain:Name=myserver,Type=WebServer,Server
=myserver
wls:/mydomain/edit !>
```

Note: `getMBean` does not throw an exception when an instance is not found.

Alternatively, the `getPath` command returns the MBean path for a specified MBean instance or `ObjectName` for the MBean in the current MBean hierarchy. See "getPath" on page A-79.

```
wls:/mydomain/serverConfig>path=getPath('com.bea:Name=myserver,Type=Server
')
wls:/mydomain/serverConfig> print path
Servers/myserver
```

Accessing Custom MBeans

WebLogic Server provides hundreds of MBeans, many of which you use to configure and monitor EJBs, Web applications, and other deployable J2EE modules. If you want to use additional MBeans to configure your applications or resources, you can create and register your own MBeans within the MBean Server subsystem on the Administration Server.

For more information on custom MBeans, see "Instrumenting and Registering Custom MBeans" in *Developing Manageable Applications with JMX*.

To navigate custom MBeans, enter the `custom` command when WLST is connected to an Administration Server or a Managed Server instance. When connected to a WebLogic Integration or WebLogic Portal server, WLST can interact with all the WebLogic Integration or WebLogic Portal Server MBeans; non-WebLogic Server MBeans appear as custom MBeans.

WLST navigates, interrogates, and edits custom MBeans as it does configuration MBeans; however, custom MBeans cannot use the `cmo` variable because a stub is not available.

Custom MBeans are editable, but not subjected to the WebLogic Server change management process. You can use MBean get and set methods, invoke, and create and delete methods on them without first entering the `startEdit` command. See "Editing Configuration MBeans" on page 4-12.

Custom MBeans are listed by domain, as shown in the following example.

```
wls:/mydomain/serverConfig> custom()
Location changed to custom tree. This is a writable tree with No root. For
more help, use help('custom')
wls:/mydomain/custom> ls()
drw-   domain1
drw-   domain2
drw-   domain3
drw-   domain4
wls:/mydomain/custom> cd("domain2")
wls:/mydomain/custom/domain2> ls()
drw-   domain2:y1=x
drw-   domain2:y2=x
wls:/mydomain/custom/domain2> cd("domain2:y1=x")
wls:/mydomain/custom/domain2/domain2:y1=x> ls()
-rw-   MyAttribute      10
wls:/mydomain/custom/domain2/domain2:y1=x>
```

Editing Configuration MBeans

Within the Administration Server, there is a set of configuration MBeans in a single, editable hierarchy whose root is `DomainMBean`. This hierarchy contains an editable copy of all configuration MBeans in the domain and it is used only as part of the change management process. The **change management process** is a controlled procedure for distributing configuration changes in a domain; a change process that loosely resembles a database transaction.

You start the editing process by obtaining a lock on the editable configuration hierarchy to prevent other people from making changes. When you finish making changes, you save and distribute them to all server instances in the domain. When you distribute changes, each server determines whether it can accept the change. If all servers are able to accept the change, they update their working hierarchy of configuration MBeans and the change is completed.

Note: The configuration lock does not prevent two processes from starting an edit session under the same user identity. BEA Systems recommends against this because when one of the sessions activates their changes, it releases the lock and the other session will not be able to save or activate their changes.

For example, if you start an edit session from WLST and start another concurrent edit session from the Administration Console under the same user name, when you save and activate your changes using WLST, you will lose the edits you are making with the Administration Console.

For more information on making and managing configuration changes, see "Configuration Change Management Process" in *Understanding Domain Configuration* and "Managing a Domain's Configuration with JMX" in *Developing Custom Management Utilities with JMX*.

Making Configuration Changes: Main Steps

The following basic steps describe the configuration editing process using WLST online:

1. To start the change process, enter the `edit` command. See "edit" on page A-123.

   This places WLST at the root of the editable configuration hierarchy and obtains an exclusive configuration lock.

2. Enter the `startEdit` command.

   The `startEdit` command initiates modifications that are treated as a part of a batch change that is not committed to the repository until you enter the `save` command.

The `startEdit` command must be called prior to invoking any command to modify the domain configuration. However, if WLST detects that there is an edit session that is already in progress by the same user, which might have been started via the Administration Console or another WLST session, it continues your edit session and you need not enter the `startEdit` command. For detailed information about `startEdit` command arguments, see "startEdit" on page A-62.

To avoid the possibility that the configuration is locked indefinitely, you can specify a time-out period. Alternatively, an administrator can enter the `cancelEdit` command to cancel an edit session and release the lock. See "cancelEdit" on page A-46.

To indicate that configuration changes are in process, an exclamation point (!) appears at the end of the WLST command prompt.

3. Use WLST edit commands to create, get and set values for, invoke operations on, and delete instances of configuration MBeans. See "Editing Commands" on page A-39.

   Use the `validate` command to ensure that changes you make are valid before saving them. See "validate" on page A-68.

4. When you finish making changes, enter the `save` command.

   The `save` command saves your changes to a pending version of the `config.xml` file; it does not release the lock. See "save" on page A-56.

5. You can make additional changes, without re-entering the `startEdit` command, or undo changes you have made by entering the `undo` command.

   The `undo` command reverts all changes that have not been saved. See "undo" on page A-67.

6. When you are ready to distribute your changes to the working configuration MBeans, enter the `activate` command.

   The `activate` command initiates the distribution of the changes and releases the lock; the exclamation point is removed from the command prompt. See "activate" on page A-41.

7. Alternatively, you can abandon your changes by entering the `stopEdit` or the `cancelEdit` command.

   The `stopEdit` command stops the current editing session and releases the edit lock. This command lets you discard any changes you made since last entering the `save` command. See "stopEdit" on page A-63.

   The `cancelEdit` command also stops the editing session and releases the configuration lock, discarding changes made since the last `save` command. However, the user entering

this command does not have to be the current editor; this allows an administrator to cancel an edit session. See "cancelEdit" on page A-46.

The WLST online script in Listing 4-5 connects WLST to an Administration Server, initiates an edit session that creates a Managed Server, saves and activates the change, initiates another edit session, creates a startup class, and targets it to the newly created server.

Start WebLogic Server before running this script. See "Running Scripts" on page 2-10.

Listing 4-5 Creating a Managed Server

```
connect("username","password")
edit()
startEdit()
svr = cmo.createServer("managedServer")
svr.setListenPort(8001)
svr.setListenAddress("my-address")
save()
activate(block="true")

startEdit()
sc = cmo.createStartupClass("my-startupClass")
sc.setClassName("com.bea.foo.bar")
sc.setArguments("foo bar")

# get the server mbean to target it

tBean = getMBean("Servers/managedServer")
if tBean != None:
    print "Found our target"
    sc.addTarget(tBean)
save()
activate(block="true")
disconnect()
exit()
```

If you attempt to make changes without first entering the `edit` command, WLST displays a message stating that you have not locked the configuration for changes and gives you the opportunity to do so. If you forget to call `save` after entering the `startEdit` command and attempt to exit the scripting shell, you are warned about the outstanding, non-committed changes. At that point you can commit or abandon all changes that you made to the configuration.

To determine if a change you made to an MBean attribute requires you to re-start the server, enter the `isRestartRequired` command. If you enter the command during an edit session, before activating your changes, it will show the attribute changes in progress that will require you to re-start the server. If you enter the command after activating your changes, it displays the attribute changes that occurred that require you to re-start the server. See "isRestartRequired" on page A-54.

Managing Configuration Changes

WebLogic Server provides a Configuration Manager service to manage the change process. The `getConfigManager` function returns the `ConfigurationManagerMBean` (see "getConfigManager" on page A-77). `ConfigurationManagerMBean` provides methods to start and stop edit sessions, and save, undo, and activate configuration changes. You use `ConfigurationManagerMBean` methods to manage configuration changes across a domain. For detailed information, see ConfigurationManagerMBean in the *WebLogic Server MBean Reference*.

The WLST online script in Listing 4-6 connects WLST to a server instance as an administrator, checks if the current editor making changes is a particular operator, then cancels the configuration edits. The script also purges all the completed activation tasks.

Start WebLogic Server before running this script. See "Running Scripts" on page 2-10.

Listing 4-6 Using the Configuration Manager

```
connect("theAdministrator","weblogic")
cmgr = getConfigManager()
user = cmgr.getCurrentEditor()
if user == "operatorSam":
    cmgr.undo()
    cmgr.cancelEdit()
cmgr.purgeCompletedActivationTasks()
```

Tracking Configuration Changes

For all changes that are initiated by WLST, you can use the `showChanges` command which displays all the changes that you made to the current configuration from the start of the edit session, including any MBean operations that were implicitly performed by the server. See Listing 4-7.

Start WebLogic Server before running this script. See "Running Scripts" on page 2-10.

Listing 4-7 Displaying Changes

```
wls:/offline> connect('username','password')
wls:/mydomain/serverConfig> edit()
wls:/mydomain/edit> startEdit()
Starting an edit session ...
wls:/mydomain/edit !> cmo.createServer('managed2')
[MBeanServerInvocationHandler]mydomain:Name=managed2,Type=Server
wls:/mydomain/edit !> cd('Servers/managed2')
wls:/mydomain/edit/Servers/managed2 !> cmo.setListenPort(7702)
wls:/mydomain/edit/Servers/managed2 !> showChanges()
Changes that are in memory and saved to disc but not yet activated are:

MBean Changed           : mydomain:Name=mydomain,Type=Domain
Operation Invoked       : add
Attribute Modified      : Servers
Attributes Old Value    : null
Attributes New Value    : managed2
Server Restart Required : false

MBean Changed           : mydomain:Name=managed2,Type=Server
Operation Invoked       : modify
Attribute Modified      : StagingDirectoryName
Attributes Old Value    : null
Attributes New Value    : .\managed2\stage
Server Restart Required : true

MBean Changed           : mydomain:Name=managed2,Type=Server
Operation Invoked       : modify
Attribute Modified      : Name
Attributes Old Value    : null
Attributes New Value    : managed2
Server Restart Required : true

MBean Changed           : mydomain:Name=managed2,Type=Server
Operation Invoked       : modify
Attribute Modified      : ListenPort
Attributes Old Value    : null
```

```
     Attributes New Value    : 7702
     Server Restart Required : false

wls:/mydomain/edit/Servers/managed2 !> save()
wls:/mydomain/edit !> activate()
Started the activation of all your changes.
The edit lock associated with this edit session is released once the activation
is successful.

The Activation task for your changes is assigned to the variable 'activationTask'

You can call the getUser() or getStatusByServer() methods on this variable to
determine the status of your activation

[MBeanServerInvocationHandler]mydomain:Type=ActivationTask
wls:/mydomain/edit/Servers/managed2>
```

The `getActivationTask` function provides information about the activation request and returns the latest `ActivationTaskMBean` which reflects the state of changes that a user is currently making or made recently. You invoke the methods that this interface provides to get information about the latest activation task in progress or just completed. For detailed information, see ActivationTaskMBean in the *WebLogic Server MBean Reference*.

The WLST online script in Listing 4-8 connects WLST to a server instance as an administrator, gets the activation task, and prints the user and the status of the task. It also prints all the changes that took place.

Start WebLogic Server before running this script. See "Running Scripts" on page 2-10.

Listing 4-8 Checking the Activation Task

```
connect("theAdministrator","weblogic")
at = getActivationTask()
print "The user for this Task "+at.getUser()+" and the state is "+at.getState()
changes = at.getChanges()
for i in changes:
    i.toString()
```

CHAPTER 5

Managing Servers and Server Life Cycle

The following sections describe how to start and stop WebLogic Server instances and monitor and manage the server life cycle using WebLogic Scripting Tool (WLST):

Managing the Server Life Cycle

During its lifetime, a server can transition through a number of operational states, such as shutdown, starting, standby, admin, resuming, and running. WLST commands such as start, suspend, resume, and shutdown cause specific changes to the state of a server instance.

Using WLST, you can:

- Start an Administration Server, with or without Node Manager. See "Starting and Stopping Servers" on page 5-2.
- Use WLST as a Node Manager client for starting, suspending, and stopping servers remotely. See "Using WLST and Node Manager to Manage Servers" on page 5-4.
- Retrieve information about the runtime state of WebLogic Server instances. See "Monitoring Server State" on page 5-6.

- Manage the life cycle of a server instance; for example, control the states through which a server instance transitions. See "Managing Server State" on page 5-7.

For more information about the server life cycle and managing servers, see "Understanding Server Life Cycle" and "Using Node Manager to Control Servers" in *Managing Server Startup and Shutdown*.

Starting and Stopping Servers

WebLogic Server provides several ways to start and stop server instances. The method that you choose depends on whether you prefer using a graphical or command-line interface, and on whether you are using the Node Manager to manage a server's life cycle.

For an overview of methods for starting and stopping server instances, see "Starting and Stopping Servers" in *Managing Server Startup and Shutdown*.

Starting an Administration Server Without Node Manager

To start an Administration Server without using Node Manager:

1. If you have not already done so, use WLST to create a domain.

   For more information, see "Creating and Configuring WebLogic Domains Using WLST Offline" on page 3-1.

2. Open a shell (command prompt) on the computer on which you created the domain.
3. Change to the directory in which you located the domain.

   By default, this directory is *BEA_HOME*\user_projects\domains*domain_name*, where *BEA_HOME* is the top-level installation directory of BEA products.

4. Set up your environment by running one of the following scripts:

   - `bin\setDomainEnv.cmd` (Windows)
   - `bin/setDomainEnv.sh` (UNIX)

   On Windows, you can use a shortcut on the Start menu to set your environment variables and invoke WLST (Tools→WebLogic Scripting Tool).

5. Invoke WLST by entering: `java weblogic.WLST`

   The WLST prompt appears.

   ```
   wls:/(offline)>
   ```

6. Use the WLST `startServer` command to start the Administration Server.

   ```
   startServer([adminServerName], [domainName], [url], [username],
   [password],[domainDir], [block], [timeout], [serverLog],
   [systemProperties], [jvmArgs])
   ```

   For detailed information about `startServer` command arguments, see "startServer" on page A-101.

   For example,

   ```
   wls:offline/>startServer('AdminServer','mydomain','t3://localhost:7001'
   ,'weblogic','weblogic','c:/bea/user_projects/domains/mydomain','true','
   60000','false')
   ```

This command starts the Administration Server without using Node Manager. However, if you use Node Manager to start the Administration Server, Node Manager supports starting, stopping, and restarting it if it fails. See "Using WLST and Node Manager to Manage Servers" on page 5-4.

After WLST starts a server instance, the server runs in a separate process from WLST; exiting WLST does not shut down the server.

Starting Managed Servers and Clusters With Node Manager

To start Managed Servers and clusters using Node Manager:

1. Invoke WLST and start an Administration Server, as described in "Starting an Administration Server Without Node Manager" on page 5-2.

2. Start Node Manager.

   The WebLogic Server custom installation process optionally installs and starts Node Manager as a Windows service on Windows systems. See "About Node Manager Installation as a Windows Service" in the *Installation Guide*. For detailed instructions, see "Starting and Running Node Manager" in *Managing Server Startup and Shutdown*.

   On Windows you can use a shortcut on the Start menu to start the Node Manager (Tools → Node Manager).

   If it's not already running, you can start Node Manager manually at a command prompt by invoking WLST and entering the `startNodeManager` command:

   ```
   c:\>java weblogic.WLST
   wls:/offline> startNodeManager()
   ```

   For more information about `startNodeManager`, see "startNodeManager" on page A-115.

3. Invoke and connect WLST to a running WebLogic Administration Server instance using the `connect` command.

```
c:\>java weblogic.WLST
wls:/(offline)> connect('username','password')
Connecting to weblogic server instance running at t3://localhost:7001
as username weblogic ...
Successfully connected to Admin Server 'myserver' that belongs to
domain 'mydomain'.
Warning: An insecure protocol was used to connect to the server.
To ensure on-the-wire security, the SSL port or Admin port should be
used instead.

wls:/mydomain/serverConfig>
```

   For detailed information about `connect` command arguments, see "connect" on page A-10.

4. Start a Managed Server instance by entering the `start` command.

```
wls:/mydomain/serverConfig>
start('managedServerName','Server','managedServerURL')
```

   For example,

```
start('managed1','Server','t3://localhost:7701')
```

5. For each Managed Server in the Administration Server's domain that you want to start, repeat step 4.

   The `start` command starts Managed Servers or clusters in a domain using Node Manager. To use the `start` command, WLST must be connected to a running Administration Server. To start Managed Servers without requiring a running Administration Server, use the `nmStart` command with WLST connected to Node Manager. See "Using WLST and Node Manager to Manage Servers" on page 5-4.

   To start clusters,

```
wls:/mydomain/serverConfig> start('mycluster', 'Cluster')
Starting the following servers in Cluster, mycluster: MS1, MS2, MS3...
.....................................................................
All servers in the cluster mycluster are started successfully.
wls:/mydomain/serverConfig>
```

   For more information, see "start" on page A-100.

Using WLST and Node Manager to Manage Servers

Node Manager is a utility for the remote control of WebLogic Server instances that lets you monitor, start, and stop server instances—both Administration Servers and Managed Servers—

and automatically restart them after a failure. For more information about Node Manager, see "Using Node Manager to Control Servers" in *Managing Server Startup and Shutdown*.

You can start, stop, and restart server instances remotely or locally, using WLST as a Node Manager client. In addition, WLST can obtain server status and retrieve the contents of the server output log.

You connect WLST to a running Node Manager instance in order to invoke Node Manager supported commands. Node Manager commands issued via WLST are processed by the Node Manager on the system hosting the target server instances. After being authenticated by Node Manager, you need not re-authenticate each time you enter a Node Manager command.

In addition, you can enroll the machine on which WLST is running to be monitored by Node Manager by entering the `nmEnroll` command. You must run this command once per domain per machine unless that domain shares the root directory of the Administration Server. WLST must be connected to an Administration Server to run this command; WLST does not need to be connected to the Node Manager. See "nmEnroll" on page A-108.

Communications from WLST to the Node Manager process on a machine include:

- Life cycle commands
- Commands to determine the availability of the Node Manager process and the health state of the server instances under Node Manager control
- Requests for log files

The following example uses WLST Node Manager commands to start, monitor, and stop an Administration Server.

1. Invoke WLST.

   ```
   java weblogic.WLST
   ```

2. Start Node Manager. (See step 2. in "Starting Managed Servers and Clusters With Node Manager")

3. Connect WLST to Node Manager by entering the `nmConnect` command.

   ```
   wls:/offline>nmConnect('username','password','nmHost','nmPort','domainName','domainDir','nmType')
   ```

   For example,

   ```
   nmConnect('weblogic', 'weblogic', 'localhost', '5556',
   'mydomain','c:/bea/user_projects/domains/mydomain','ssl')
   ```

```
Connecting to Node Manager ...
Successfully connected.
wls:/nm/mydomain>
```

For detailed information about `nmConnect` command arguments, see "nmConnect" on page A-106.

After successfully connecting WLST to Node Manager, you can start, monitor, and stop Administration and Managed Server instances.

When connected to Node Manager, the `nmStart` command starts Managed Servers without requiring a running Administration Server.

4. Use the `nmStart` command to start an Administration Server.

```
wls:/nm/mydomain>nmStart('serverName')
starting server AdminServer
...
Server AdminServer started successfully
wls:/nm/mydomain>
```

5. Monitor the status of the server you started by entering the `nmServerStatus` command.

```
wls:/nm/mydomain>nmServerStatus('serverName')
RUNNING
wls:/nm/mydomain>
```

6. Stop the server by entering the `nmKill` command.

```
wls:/nm/mydomain>nmKill('serverName')
Killing server AdminServer
Server AdminServer killed successfully
wls:/nm/mydomain>
```

For more information about WLST Node Manager commands, see "Node Manager Commands" on page A-104.

Monitoring Server State

WebLogic Server displays and stores information about the current operational state of a server instance and state transitions that have occurred since the server instance started up. This information is useful to administrators who:

- Monitor the availability of server instances and the applications they host.
- Perform day-to-day operations tasks, including startup and shutdown procedures.

- Plan correction actions, such as migration of services, when a server instance fails or crashes.

Using WLST, you can obtain the state of a server instance in the following ways:

- Use the `state` command—returns the state of a server or cluster.

```
wls:/mydomain/serverConfig> state('serverName','Server')
Current state of 'managed1' : RUNNING
wls:/mydomain/serverConfig>
```

  See "state" on page A-88.

- Navigate to the `ServerRuntimeMBean` and display the `State` attribute.

```
wls:/mydomain/serverConfig> serverRuntime()
wls:/mydomain/serverRuntime> ls()
-r--   State      RUNNING
```

To tailor WLST server monitoring, shutdown, and restart behaviors, see "Script for Monitoring Server State" on page 6-11.

Managing Server State

WLST life cycle commands let you control the states through which a server instance transitions. See "Life Cycle Commands" on page A-94.

The commands in Listing 5-1 explicitly move WebLogic Server through the following server states: `RUNNING->ADMIN->RUNNING->SHUTDOWN`.

Start WebLogic Server before running this script. See "Running Scripts" on page 2-10.

Listing 5-1 WLST Life Cycle Commands

```
connect("username","password","t3://localhost:8001")

# First enable the Administration Port. This is Not a requirement.
edit()
startEdit()
cmo.setAdministrationPortEnabled(1)
activate(block="true")

# check the state of the server
state("myserver")

# now move the server from RUNNING state to ADMIN
suspend("myserver", block="true")
```

```
# check the state
state("myserver")

# now resume the server to RUNNING state
resume("myserver",block="true")

# check the state
state("myserver")

# now take a thread dump of the server
threadDump("./dumps/threadDumpAdminServer.txt")

# finally shutdown the server
shutdown(block="true")
```

CHAPTER 6

Automating WebLogic Server Administration Tasks

You can use the WebLogic Scripting Tool (WLST) to automate the creation and management of WebLogic Server domains, servers, and resources. WLST provides commands that create, get and set values for, invoke operations on, and delete instances of configuration MBeans and commands to get values and invoke operations on runtime MBeans. The following sections describe using WLST commands online to automate typical domain and server configuration tasks:

- "Creating a Sample Domain: Main Steps" on page 6-2
- "Monitoring Domain Runtime Information" on page 6-10
- "Managing Security" on page 6-13
- "Configuring Logging" on page 6-20

Alternatively, you can use one of the following techniques to automate the configuration of a WebLogic Server domain:

- Use WLST offline to create a new domain or update an existing domain without connecting to a running WebLogic Server. WLST offline supports the same functionality as the Configuration Wizard. See "Creating and Configuring WebLogic Domains Using WLST Offline" on page 3-1.
- Use the WebLogic Server Ant tasks. For almost all configuration needs, the Ant tasks and the `weblogic.Server`, `weblogic.Admin` (deprecated in this release of WebLogic Server), and `weblogic.Deployer` commands are functionally equivalent. See "Using Ant Tasks to Configure and Use a WebLogic Server Domain" in *Developing Applications with WebLogic Server.*

Creating a Sample Domain: Main Steps

The section "Script to Create and Configure a Sample Domain" on page 6-7 provides a sample script for creating a modified MedRec domain. The script creates a new directory `MedRecDomain` under the current directory, and creates and starts an Administration Server. WLST connects to the server and builds a modified version of the MedRec domain.

The sample script does the following:

1. Creates a new directory, `MedRecDomain`, under the current directory.
2. Creates and starts an Administration Server. See "Creating a Domain" on page 6-3.
3. After the domain's Administration Server has completed its startup cycle, connects WLST to the server and configures resources for the domain. See:
   - "Creating JDBC Resources" on page 6-3
   - "Creating JMS Resources" on page 6-4
   - "Creating Mail Resources" on page 6-5
4. Invokes multiple WLST `deploy` commands to deploy J2EE modules such as EJBs and Enterprise applications. See "Deploying Applications" on page 6-6.

Setting Up the Environment

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

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

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

where *`WL_HOME`* refers to the top-level installation directory for WebLogic Server.

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

`WL_HOME``\samples\domains\wl_server\setExamplesEnv.cmd` (on Windows)
`WL_HOME``/samples/domains/wl_server/setExamplesEnv.sh` (on UNIX)

Creating a Domain

The commands in Listing 6-1 create a domain named MedRecDomain with an Administration Server named medrec-adminServer that listens on port 8001, and connect WLST to the server instance.

Listing 6-1 Creating a Domain

```
from java.io import File
domainDir = File("MedRecDomain")
bool = domainDir.mkdir()
if bool==1:
  print 'Successfully created a new Directory'
else:
  if domainDir.delete()==1:
    domainDir.mkdir()
    print 'Successfully created a new Directory'
  else:
    print 'Could not create new directory, dir already exists'
    stopExecution("cannot create a new directory")

debug()
dsname="myJDBCDataSource"
adminServerName="medrec-adminServer"
domainName="MedRecDomain"
_url="t3://localhost:8001"
uname="weblogic"
pwd="weblogic"
startNewServer(adminServerName,domainName,_url,domainDir=domainDir.getPath(),
block="true")
connect(uname, pwd, _url)
edit()
startEdit()
```

Note: The command specifies a listen port of 8001 because the sample MedRec domain that WebLogic Server installs listens on the default port 7011. If the sample MedRec domain is running, the 7011 listen port cannot be used by another server instance.

Creating JDBC Resources

The commands in Listing 6-2 create and configure a JDBC system resource for MedRecDomain.

Listing 6-2 Creating a JDBC System Resource

```
dsname="myJDBCDataSource"

# Creating and Configuring a JDBC System Resource
print 'Creating JDBCSystemResource with name '+dsname
jdbcSR = create(dsname,"JDBCSystemResource")
theJDBCResource = jdbcSR.getJDBCResource()
theJDBCResource.setName("myJDBCDataSource")

connectionPoolParams = theJDBCResource.getJDBCConnectionPoolParams()
connectionPoolParams.setConnectionReserveTimeoutSeconds(25)
connectionPoolParams.setMaxCapacity(100)
connectionPoolParams.setTestTableName("SYSTABLES")

dsParams = theJDBCResource.getJDBCDataSourceParams()
dsParams.addJNDIName("ds.myJDBCDataSource")

driverParams = theJDBCResource.getJDBCDriverParams()
driverParams.setUrl("jdbc:pointbase:server://localhost/demo")
driverParams.setDriverName("com.pointbase.xa.xaDataSource")
#driverParams.setUrl("jdbc:oracle:thin:@my-oracle-server:my-oracle-server-port
:my-oracle-sid")
#driverParams.setDriverName("oracle.jdbc.driver.OracleDriver")

driverParams.setPassword("examples")
#driverParams.setLoginDelaySeconds(60)
driverProperties = driverParams.getProperties()

proper = driverProperties.createProperty("user")
#proper.setName("user")
proper.setValue("examples")

proper1 = driverProperties.createProperty("DatabaseName")
#proper1.setName("DatabaseName")
proper1.setValue("jdbc:pointbase:server://localhost/demo")
```

Creating JMS Resources

The commands in Listing 6-3 create a JMS system resource in MedRecDomain.

Listing 6-3 Creating a JMS System Resource

```
# Creating a JMS System Resource
jmsSystemResource = create("medrec-jms-resource","JMSSystemResource")
theJMSResource = jmsSystemResource.getJMSResource()

# Creating a JMS Connection Factory
mrqFactory = theJMSResource.createConnectionFactory("MedRecQueueFactory")
mrqFactory.setJNDIName("jms/MedRecQueueConnectionFactory")

# Creating and Configuring a JMS JDBC Store
mrjStore = create("MedRecJMSJDBCStore","JDBCStore")
mrjStore.setDataSource(jdbcSR)
mrjStore.setPrefixName("MedRec")

# Creating and Configuring a JMS Server
mrJMSServer = create("MedRecJMSServer","JMSServer")
mrJMSServer.setStore(mrjStore)

# Creating and Configuring a Queue
regQueue = theJMSResource.createQueue("RegistrationQueue")
regQueue.setJNDIName("jms/REGISTRATION_MDB_QUEUE")

# Creating and Configuring an Additional Queue
mailQueue = theJMSResource.createQueue("MailQueue")
mailQueue.setJNDIName("jms/MAIL_MDB_QUEUE")
```

See "Using the WebLogic Scripting Tool to Manage JMS Servers and JMS System Resources" in *Configuring and Managing WebLogic JMS*.

Creating Mail Resources

The commands in Listing 6-4 add E-mail capabilities to the sample applications in MedRecDomain by creating and configuring a `MailSessionMBean`.

Listing 6-4 Creating Mail Resources

```
# Creating Mail Resources
mrMailSession = create("MedicalRecordsMailSession","MailSession")
mrMailSession.setJNDIName("mail/MedRecMailSession")
mrMailSession.setProperties(makePropertiesObject("mail.user=joe;mail.host=mail
.mycompany.com"))
```

To see all attributes and legal values of the `MailSessionMBean`, see `MailSessionMBean` in the *WebLogic Server MBean Reference*. For more information about the WebLogic Server mail service, see "Configure Access to JavaMail" in the *Administration Console Online Help*.

Deploying Applications

The commands in Listing 6-5 deploy sample applications in MedRecDomain.

Listing 6-5 Deploying Applications

```
# Deploying Applications

deploy("PhysicianEAR","C:/bea/weblogic91/samples/server/medrec/src/physicianEa
r","medrec-adminServer",securityModel="Advanced",block="true")
deploy("StartupEAR","C:/bea/weblogic91/samples/server/medrec/src/startupEar","
medrec-adminServer",securityModel="Advanced",block="true")
deploy("MedRecEAR","C:/bea/weblogic91/samples/server/medrec/src/medrecEAR","me
drec-adminServer",securityModel="Advanced",block="true")
```

Notes:

- You must invoke these commands on the computer that hosts the Administration Server for MedRecDomain.
- Because the sample applications use JDBC connection pools, you must specify the JDBC driver in the `CLASSPATH` environment variable. See "Setting Up the Environment" on page 6-2.
- To use JDBC resources in MedRecDomain, you must start the database before running the script.
- In the sample commands, the application files are located in the *`WL_HOME`*`\samples\server\medrec\src` directory.

For more information using WLST for deploying applications, see "Overview of Deployment Tools" in *Deploying Applications to WebLogic Server.*

Script to Create and Configure a Sample Domain

Listing 6-6 provides a sample script for creating a modified MedRec domain. You can use the script as a template for creating and configuring a typical WebLogic Server domain.

Note: This sample script uses the demonstration PointBase Server that is installed with WebLogic Server. Before running the script, you should start the PointBase Server by issuing one of the following commands:

Windows: *WL_HOME*\common\eval\pointbase\tools\startPointBase.cmd

UNIX: *WL_HOME*/common/eval/pointbase/tools/startPointBase.sh

To create and configure a domain such as the MedRec sample domain:

1. Copy and save the commands in Listing 6-6 in a text file with a .py file extension; for example, `cloneDomain.py`.
2. Set the required environment variables. See "Setting Up the Environment" on page 6-2.

   Use *WL_HOME*\samples\domains\wl_server\setExamplesEnv.cmd to set the variables required for running this script.
3. Invoke WLST and run the sample script by entering the following command:

   `java weblogic.WLST <filepath>/cloneDomain.py`

   For more information on how to run this script, see "Running Scripts" on page 2-10.

Listing 6-6 cloneDomain.py

```
from java.io import File
domainDir = File("MedRecDomain")
bool = domainDir.mkdir()
if bool==1:
  print 'Successfully created a new Directory'
else:
  if domainDir.delete()==1:
    domainDir.mkdir()
    print 'Successfully created a new Directory'
  else:
    print 'Could not create new directory, dir already exists'
    stopExecution("cannot create a new directory")

debug()
dsname="myJDBCDataSource"
adminServerName="medrec-adminServer"
```

```
domainName="MedRecDomain"
_url="t3://localhost:8001"
uname="weblogic"
pwd="weblogic"
startNewServer(adminServerName,domainName,_url,domainDir=domainDir.getPath(),
block="true")
connect(uname, pwd, _url)
edit()
startEdit()

# Creating and Configuring a JDBC System Resource
print 'Creating JDBCSystemResource with name '+dsname
jdbcSR = create(dsname,"JDBCSystemResource")
theJDBCResource = jdbcSR.getJDBCResource()
theJDBCResource.setName("myJDBCDataSource")

connectionPoolParams = theJDBCResource.getJDBCConnectionPoolParams()
connectionPoolParams.setConnectionReserveTimeoutSeconds(25)
connectionPoolParams.setMaxCapacity(100)
connectionPoolParams.setTestTableName("SYSTABLES")

dsParams = theJDBCResource.getJDBCDataSourceParams()
dsParams.addJNDIName("ds.myJDBCDataSource")

driverParams = theJDBCResource.getJDBCDriverParams()
driverParams.setUrl("jdbc:pointbase:server://localhost/demo")
driverParams.setDriverName("com.pointbase.xa.xaDataSource")
#driverParams.setUrl("jdbc:oracle:thin:@my-oracle-server:my-oracle-server-port
:my-oracle-sid")
#driverParams.setDriverName("oracle.jdbc.driver.OracleDriver")

driverParams.setPassword("examples")
#driverParams.setLoginDelaySeconds(60)
driverProperties = driverParams.getProperties()

proper = driverProperties.createProperty("user")
#proper.setName("user")
proper.setValue("examples")

proper1 = driverProperties.createProperty("DatabaseName")
#proper1.setName("DatabaseName")
proper1.setValue("jdbc:pointbase:server://localhost/demo")

# Creating a JMS System Resource
jmsSystemResource = create("medrec-jms-resource","JMSSystemResource")
theJMSResource = jmsSystemResource.getJMSResource()

# Creating a JMS Connection Factory
mrqFactory = theJMSResource.createConnectionFactory("MedRecQueueFactory")
mrqFactory.setJNDIName("jms/MedRecQueueConnectionFactory")
```

```
# Creating and Configuring a JMS JDBC Store
mrjStore = create("MedRecJMSJDBCStore","JDBCStore")
mrjStore.setDataSource(jdbcSR)
mrjStore.setPrefixName("MedRec")

# Creating and Configuring a JMS Server
mrJMSServer = create("MedRecJMSServer","JMSServer")
mrJMSServer.setPersistentStore(mrjStore)

# Creating and Configuring a Queue
regQueue = theJMSResource.createQueue("RegistrationQueue")
regQueue.setJNDIName("jms/REGISTRATION_MDB_QUEUE")

# Creating and Configuring an Additional Queue
mailQueue = theJMSResource.createQueue("MailQueue")
mailQueue.setJNDIName("jms/MAIL_MDB_QUEUE")

# Creating Mail Resources
mrMailSession = create("MedicalRecordsMailSession","MailSession")
mrMailSession.setJNDIName("mail/MedRecMailSession")
mrMailSession.setProperties(makePropertiesObject("mail.user=joe;mail.host=mail
.mycompany.com"))

# Getting and configuring the server target
tgt = getMBean("/Servers/medrec-adminServer")
tgt.setJavaCompiler("javac")
tgt.setListenAddress("localhost")
tgt.setListenPort(8001)
#tgt.setIIOPEnabled(0)
tgt.setInstrumentStackTraceEnabled(1)

ssl = tgt.getSSL()
ssl.setEnabled(1)
ssl.setIdentityAndTrustLocations("KeyStores")
ssl.setListenPort(9992)

# Targeting Resources to the medrec admin server
jdbcSR.addTarget(tgt)
jmsSystemResource.addTarget(tgt)
mrjStore.addTarget(tgt)
mrJMSServer.addTarget(tgt)
mrMailSession.addTarget(tgt)

save()
activate(block="true")
shutdown(force="true",block="true")
print 'end of the script ... '
```

Monitoring Domain Runtime Information

WebLogic Server includes a large number of MBeans which provide information about the runtime state of its resources. Each server instance in a domain hosts only the MBeans that configure and monitor its own set of resources. However, within the Administration Server, MBeans for domain-wide services are in a single hierarchy whose root is `DomainRuntimeMBean`. The domain runtime MBean hierarchy provides access to any runtime MBean on any server in the domain as well as MBeans for domain-wide services such as application deployment, JMS servers, and JDBC connection pools.

Accessing Domain Runtime Information: Main Steps

Accessing the runtime information for a domain includes the following main steps:

1. Invoke WLST and connect to a running Administration Server instance. See "Invoking WLST" on page 2-8.
2. Navigate to the domain runtime MBean hierarchy by entering the `domainRuntime` command.

   ```
   wls:/mydomain/serverConfig>domainRuntime()
   ```

   The `domainRuntime` command places WLST at the root of the domain-wide runtime management objects, `DomainRuntimeMBean`.
3. Navigate to `ServerRuntimes` and then to the server instance which you are interested in monitoring.

   ```
   wls:/mydomain/domainRuntime>cd('ServerRuntimes/myserver')
   ```
4. At the server instance, navigate to and interrogate runtime MBeans.

   ```
   wls:/mydomain/domainRuntime/ServerRuntimes/myserver>cd('JVMRuntime/myse
   rver')>
   wls:/mydomain/domainRuntime/ServerRuntimes/myserver/JVMRuntime/myserver
   >ls()
   -r--   AllProcessorsAverageLoad                    0.0
   -r--   Concurrent                                  true
   -r--   FreeHeap                                    15050064
   -r--   FreePhysicalMemory                          900702208
   -r--   GCHandlesCompaction                         true
   -r--   GcAlgorithm                          Dynamic GC currently running
   strategy: Nursery, parallel mark, parallel sweep
   -r--   Generational                                true
   ```

```
-r--   HeapFreeCurrent                          14742864
-r--   HeapFreePercent                          5
-r--   HeapSizeCurrent                          268435456
-r--   HeapSizeMax                              268435456
-r--   Incremental                              false
-r--   JVMDescription                         BEA JRockit Java Virtual
Machine
-r--   JavaVMVendor                             BEA Systems, Inc.
-r--   JavaVendor                               BEA Systems, Inc.
-r--   JavaVersion                              1.5.0
...
```

The following sections provide example scripts for retrieving runtime information about WebLogic Server server instances and domain resources.

Script for Monitoring Server State

The WLST online script in Listing 6-7 checks the status of a Managed Server every 5 seconds and restarts the server if the server state changes from `RUNNING` to any other status.

For information on how to run this script, see "Running Scripts" on page 2-10.

Listing 6-7 Monitoring Server State

```
# Node Manager needs to be running to run this script.

import thread
import time

def checkHealth(serverName):
  while 1:
    slBean = getSLCRT(serverName)
    status = slBean.getState()
    print 'Status of Managed Server is '+status
    if status != "RUNNING":
      print 'Starting server '+serverName
      start(serverName, block="true")
    time.sleep(5)

def getSLCRT(svrName):
    domainRuntime()
    slrBean = cmo.lookupServerLifecycleRuntime(svrName)
    return slcBean
```

Script for Monitoring the JVM

The WLST online script in Listing 6-8 monitors the `HJVMHeapSize` for all running servers in a domain; it checks the heap size every 3 minutes and prints a warning if the heap size is greater than a specified threshold.

For information on how to run this script, see "Running Scripts" on page 2-10.

Listing 6-8 Monitoring the JVM Heap Size

```
waitTime=300000
THRESHOLD=100000000
uname = "weblogic"
pwd = "weblogic"
url = "t3://localhost:7001"
def monitorJVMHeapSize():
    connect(uname, pwd, url)
    while 1:
        serverNames = getRunningServerNames()
        domainRuntime()
        for name in serverNames:
            print 'Now checking '+name.getName()
            try:
           cd("/ServerRuntimes/"+name.getName()+"/JVMRuntime/"+name.getName())
            except WLSTException,e:
                # this typically means the server is not active, just ignore
                pass
            heapSize = cmo.getHeapSizeCurrent()
            if heapSize > THRESHOLD:
            # do whatever is neccessary, send alerts, send email etc
                print 'WARNING: The HEAPSIZE is Greater than the Threshold'
            else:
                print heapSize
        java.lang.Thread.sleep(1800000)

def getRunningServerNames():
    domainConfig()
    return cmo.getServers()

if __name__== "main":
    monitorJVMHeapSize()
```

Managing Security

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

You can use WLST to invoke operations on the following types of Authentication providers:

- The default WebLogic Server Authentication provider, `weblogic.management.security.authentication.AuthenticatorMBean`. By default, all security realms use this Authentication provider to manage users and groups.
- Custom Authentication providers that extend `weblogic.security.spi.AuthenticationProvider` and extend the optional Authentication SSPI MBeans. See "SSPI MBean Quick Reference" in *Developing Security Providers for WebLogic Server*.

Note: Use the Edit MBean Server to modify security MBean attributes; use a Runtime MBean Server or the Domain Runtime MBean Server to invoke security provider MBean operations. You cannot invoke these operations if an edit is in process or if you need to reboot the server because you've modified a security MBean attribute.

For more information, see "Choosing an MBean Server to Manage Security Realms" in *Developing Custom Management Utilities with JMX*.

The following sections describe basic tasks for managing users and groups using WLST:

For information about additional tasks that the `AuthenticationProvider` and the optional MBeans support, refer to `weblogic.management.security.authentication` package in the *WebLogic Server MBean Reference*.

Note: WebLogic Server 6.0 style security MBeans are accessible using WLST but are not displayed using the `ls` command. For example, if you enter the following commands, WLST lists the domain MBeans, but not excluded attributes, such as FileRealms:

```
java weblogic.WLST
connect()
ls()
```

However, if you enter the following commands, WLST displays the `DomainMBean`'s file realms:

```
java weblogic.WLST
connect()
cmo.getFileRealms()
```

Creating a User

To create a user, invoke the `UserEditorMBean.createUser` method, which is extended by the security realm's `AuthenticationProvider` MBean. For more information, see the `createUser` method in the *WebLogic Server MBean Reference*.

The method requires three input parameters:

`username password user-description`

The following WLST online script invokes `createUser` on the default Authentication Provider. For information on how to run this script, see "Running Scripts" on page 2-10.

Listing 6-9 Creating a User

```
from weblogic.management.security.authentication import UserEditorMBean

print "Creating a user ..."
atnr=cmo.getSecurityConfiguration().getDefaultRealm().lookupAuthenticationProv
ider("DefaultAuthenticator")
atnr.createUser('my_user','my_password','new_admin')
print "Created user successfully"
```

Adding a User to a Group

To add a user to a group, invoke the `GroupEditorMBean.addMemberToGroup` method, which is extended by the security realm's `AuthenticationProvider` MBean. For more information, see the `addMemberToGroup` method in the *WebLogic Server MBean Reference*.

The method requires two input parameters:

`groupname username`

The following WLST online script invokes `addMemberToGroup` on the default Authentication Provider. For information on how to run this script, see "Running Scripts" on page 2-10.

Listing 6-10 Adding a User to a Group

```
from weblogic.management.security.authentication import GroupEditorMBean

print "Adding a user ..."
atnr=cmo.getSecurityConfiguration().getDefaultRealm().lookupAuthenticationProv
ider("DefaultAuthenticator")
atnr.addMemberToGroup('Administrators','my_user')
print "Done adding a user"
```

Verifying Whether a User Is a Member of a Group

To verify whether a user is a member of a group, invoke the `GroupEditorMBean.isMember` method, which is extended by the security realm's `AuthenticationProvider` MBean. For more information, see the `isMember` method in the *WebLogic Server MBean Reference*.

The method requires three input parameters:

`groupname username boolean`

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

The following WLST online script invokes `isMember` on the default Authentication Provider. For information on how to run this script, see "Running Scripts" on page 2-10.

Listing 6-11 Verifying Whether a User is a Member of a Group

```
from weblogic.management.security.authentication import GroupEditorMBean

print "Checking if isMember of a group ... "
atnr=cmo.getSecurityConfiguration().getDefaultRealm().lookupAuthenticationProv
ider("DefaultAuthenticator")
if atnr.isMember('Administrators','my_user',true) == 0:
  print "my_user is not member of Administrators"
else:
  print "my_user is a member of Administrators"
```

Listing Groups to Which a User Belongs

To see a list of groups that contain a user or a group, invoke the `MemberGroupListerMBean.listMemberGroups` method, which is extended by the security realm's `AuthenticationProvider` MBean. For more information, see the `listMemberGroups` method in the *WebLogic Server MBean Reference*.

The method requires one input parameter:

`memberUserOrGroupName`

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

The following WLST online script invokes `listMemberGroups` on the default Authentication provider. For information on how to run this script, see "Running Scripts" on page 2-10.

Listing 6-12 Listing Groups to Which a User Belongs

```
from weblogic.management.security.authentication import MemberGroupListerMBean

print "Listing the member groups ..."
atnr=cmo.getSecurityConfiguration().getDefaultRealm().lookupAuthenticationProv
ider("DefaultAuthenticator")
x = atnr.listMemberGroups('my_user')
print x
```

The method returns a cursor, which refers to a list of names. The `weblogic.management.utils.NameLister.haveCurrent`, `getCurrentName`, and `advance`

methods iterate through the returned list and retrieve the name to which the current cursor position refers. See the `weblogic.management.utils.NameLister` interface in the *WebLogic Server MBean Reference*.

Listing Users and Groups in a Security Realm

To see a list of user or group names, you invoke a series of methods, all of which are available through the `AuthenticationProvider` interface:

- The `GroupReaderMBean.listGroups` and `UserReaderMBean.listUsers` methods take two input parameters: a pattern of user or group names to search for, and the maximum number of names that you want to retrieve.

  Because a security realm can contain thousands (or more) of user and group names that match the pattern, the methods return a cursor, which refers to a list of names.

  For more information, see the `listGroups` and `listUsers` methods in the *WebLogic Server MBean Reference*.

- The `NameLister.haveCurrent`, `getCurrentName`, and `advance` methods iterate through the returned list and retrieve the name to which the current cursor position refers. For more information, see the `NameListerMBean` interface in the *WebLogic Server MBean Reference*.

- The `NameLister.close` method releases any server-side resources that are held on behalf of the list.

The WLST online script in Listing 6-13 lists all the users in a realm and the groups to which they belong. For information on how to run this script, see "Running Scripts" on page 2-10.

Listing 6-13 Listing Users and Groups

```
from weblogic.management.security.authentication import UserReaderMBean
from weblogic.management.security.authentication import GroupReaderMBean

realm=cmo.getSecurityConfiguration().getDefaultRealm()
atns = realm.getAuthenticationProviders()
for i in atns:
  if isinstance(i,UserReaderMBean):
    userReader = i
    cursor = i.listUsers("*",0)
    print 'Users in realm '+realm.getName()+' are: '
    while userReader.haveCurrent(cursor):
      print userReader.getCurrentName(cursor)
```

```
      userReader.advance(cursor)
    userReader.close(cursor)

for i in atns:
  if isinstance(i,GroupReaderMBean):
    groupReader = i
    cursor = i.listGroups("*",0)
    print 'Groups in realm are: '
    while groupReader.haveCurrent(cursor):
      print groupReader.getCurrentName(cursor)
      groupReader.advance(cursor)
    groupReader.close(cursor)
```

Changing a Password

To change a user's password, invoke the `UserPasswordEditorMBean.changeUserPassword` method, which is extended by the security realm's `AuthenticationProvider` MBean. For more information, see the `changeUserPassword` method in the *WebLogic Server MBean Reference*.

The following WLST online script invokes `changeUserPassword` on the default Authentication Provider: For information on how to run this script, see "Running Scripts" on page 2-10.

Listing 6-14 Changing a Password

```
from weblogic.management.security.authentication import UserPasswordEditorMBean

print "Changing password ..."
atnr=cmo.getSecurityConfiguration().getDefaultRealm().lookupAuthenticationProv
ider("DefaultAuthenticator")
  atnr.changeUserPassword('my_user','my_password','new_password')
print "Changed password successfully"
```

Protecting User Accounts in a Security Realm

WebLogic Server provides a set of attributes to protect user accounts from intruders. By default, these attributes are set for maximum protection. You can decrease the level of protection for user accounts. For example, you can increase the number of login attempts before a user account is

locked, increase the time period in which invalid login attempts are made before locking the user account, or change the amount of time a user account is locked.

The `AuthenticationProvider` MBean does not extend methods that you use to protect user accounts. Instead, retrieve the `UserLockoutManagerMBean` and invoke its methods. For more information, see the `UserLockoutManagerMBean` interface in the *WebLogic Server MBean Reference*.

The following tasks provide examples for invoking `UserLockoutManagerMBean` methods:

- "Set Consecutive Invalid Login Attempts" on page 6-19
- "Unlock a User Account" on page 6-20

Set Consecutive Invalid Login Attempts

The following WLST online script sets the number of consecutive invalid login attempts before a user account is locked out. For information on how to run this script, see "Running Scripts" on page 2-10.

Listing 6-15 Setting Consecutive Invalid Login Attempts

```
from weblogic.management.security.authentication import UserLockoutManagerMBean

edit()
startEdit()

#You have two choices for getting a user lockout manager to configure
# 1 - to configure the default realm's UserLockoutManager:

ulm=cmo.getSecurityConfiguration().getDefaultRealm().getUserLockoutManager()

# 2 - to configure another realm's UserLockoutManager:
#ulm=cmo.getSecurityConfiguration().lookupRealm("anotherRealm").getUserLockout
Manager()

ulm.setLockoutThreshold(3)
save()
activate()
```

Unlock a User Account

The following WLST online script unlocks a user account. For information on how to run this script, see "Running Scripts" on page 2-10.

Listing 6-16 Unlocking a User Account

```
from weblogic.management.security.authentication import UserLockoutManagerMBean

serverRuntime()
ulm=cmo.getServerSecurityRuntime().getDefaultRealmRuntime().getUserLockoutMana
gerRuntime()
#note1 : You can only manage user lockouts for the default realm starting from
when the server was booted (versus other non-active realms).
#note2 : If the default realm's user lockout manager's LockoutEnabled attribute
is false, then the user lockout manager's runtime MBean will be null.
#That is, you can only manage user lockouts in the default realm if its user
lockout manager is enabled.

if ulm != None:
  ulm.clearLockout("myuser")
```

Configuring Logging

Using WLST, you can configure a server instance's logging and message output.

To determine which log attributes can be configured, see `LogMBean` and `LogFileMBean` in the *WebLogic Server MBean Reference*. The reference also indicates valid values for each attribute.

The WLST online script in Listing 6-17 get and set several `LogMBean` and `LogFileMBean` attributes. For information on how to run this script, see "Running Scripts" on page 2-10.

Listing 6-17 Configuring Logging

```
from java.lang import Boolean
from java.lang import System
from java.lang import Integer

username = System.getProperty("user","weblogic")
password = System.getProperty("password","weblogic")
adminHost = System.getProperty("adminHost","localhost")
adminPort = System.getProperty("adminPort","7001")
```

```
protocol = System.getProperty("protocol","t3")
url = protocol+"://"+adminHost+":"+adminPort

fileCount = Integer.getInteger("fileCount", 5)
fileMinSize = Integer.getInteger("fileMinSize", 400)
fileName =
System.getProperty("fileName","config\\mydomain\\myserver\\myserver.log")
fileTimeSpan = Integer.getInteger("fileTimeSpan", 12)
log4jEnabled = System.getProperty("log4jEnabled", "true")
stdoutSeverity = System.getProperty("stdoutSeverity", "Info")
logBRSeverity = System.getProperty("logBRSeverity", "Info")
logFileSeverity = System.getProperty("logFileSeverity", "Info")
memBufferSeverity = System.getProperty("memBufferSeverity", "Info")
memBufferSize = Integer.getInteger("memBufferSize", 400)
numOfFilesLimited = System.getProperty("numOfFilesLimited", "true")
redirectStdout = System.getProperty("redirectStdout", "true")
redirectStdErr = System.getProperty("redirectStdErr", "true")
rotateOnStartup = System.getProperty("rotateOnStartup", "false")
rotateTime = System.getProperty("rotateTime", "00:10")
rotateType = System.getProperty("rotateType", "byTime")

print "Connecting to " + url + " as [" + \
  username + "," + password + "]"

# Connect to the server
connect(username,password,url)
edit()
startEdit()

# set CMO to the server log config
cd("Servers/myserver/Log/myserver")
ls ()

# change the LogFileMBean and LogMBean attributes
print "Original FileCount is " + 'get("FileCount")'
print "Setting FileCount to be " + `fileCount`
set("FileCount", fileCount)

print "Original FileMinSize is " + 'get("FileMinSize")'
print "Setting FileMinSize to be " + 'fileMinSize'
set("FileMinSize", fileMinSize)

print "Original FileName is " + 'get("FileName")'
print "Setting FileName to be " + 'fileName'
set("FileName", fileName)

print "Original FileTimeSpan is " + 'get("FileTimeSpan")'
print "Setting FileTimeSpan to be " + 'fileTimeSpan'
set("FileTimeSpan", fileTimeSpan)
```

```
print "Original Log4jEnabled is " + 'get("Log4jLoggingEnabled")'
print "Setting Log4jLoggingEnabled to be " + 'log4jEnabled'
set("Log4jLoggingEnabled", log4jEnabled)

print "Original StdoutSeverity is " + 'get("StdoutSeverity")'
print "Setting StdoutSeverity to be " + 'stdoutSeverity'
set("StdoutSeverity", stdoutSeverity)

print "Original DomainLogBroadcastSeverity is " +
`get("DomainLogBroadcastSeverity")`
print "Setting DomainLogBroadcastSeverity to be " + 'logBRSeverity'
set("DomainLogBroadcastSeverity", logBRSeverity)

print "Original LogFileSeverity is " + 'get("LogFileSeverity")'
print "Setting LogFileSeverity to be " + 'logFileSeverity'
set("LogFileSeverity", logFileSeverity)

print "Original MemoryBufferSeverity is " + 'get("MemoryBufferSeverity")'
print "Setting MemoryBufferSeverity to be " + 'memBufferSeverity'
set("MemoryBufferSeverity", memBufferSeverity)

print "Original MemoryBufferSize is " + 'get("MemoryBufferSize")'
print "Setting MemoryBufferSize to be " + 'memBufferSize'
set("MemoryBufferSize", memBufferSize)

print "Original NumberOfFilesLimited is " + 'get("NumberOfFilesLimited")'
print "Setting NumberOfFilesLimited to be " + 'numOfFilesLimited'
set("NumberOfFilesLimited", numOfFilesLimited)

print "Original RedirectStdoutToServerLogEnabled is " +
'get("RedirectStdoutToServerLogEnabled")'
print "Setting RedirectStdoutToServerLogEnabled to be " + 'redirectStdout'
set("RedirectStdoutToServerLogEnabled", redirectStdout)

print "Original RedirectStderrToServerLogEnabled is " +
'get("RedirectStderrToServerLogEnabled")'
print "Setting RedirectStderrToServerLogEnabled to be " + 'redirectStdErr'
set("RedirectStderrToServerLogEnabled", redirectStdErr)

print "Original RotateLogOnStartup is " + 'get("RotateLogOnStartup")'
print "Setting RotateLogOnStartup to be " + 'rotateOnStartup'
set("RotateLogOnStartup", rotateOnStartup)

print "Original RotationTime is " + 'get("RotationTime")'
print "Setting RotationTime to be " + 'rotateTime'
set("RotationTime", rotateTime)

print "Original RotationType is " + 'get("RotationType")'
print "Setting RotationType to be " + 'rotateType'
set("RotationType", rotateType)
```

```
save()
activate()

print
ls ()

# all done...
exit()
```

For example scripts that demonstrate using WLST to configure the WebLogic Diagnostic Framework, see "WebLogic Scripting Tool Examples" in *Configuring and Using the WebLogic Diagnostics Framework*.

APPENDIX A

WLST Command and Variable Reference

The following sections describe the WLST commands and variables in detail. Topics include:

Overview of WSLT Command Categories

Note: It is recommended that you review "Requirements for Entering WLST Commands" on page 2-9 for command syntax requirements.

WLST commands are divided into the following categories.

Table A-1 WLST Command Categories

Command Category	Description
Browse Commands	Navigate the hierarchy of configuration or runtime beans and control the prompt display.
Control Commands	• Connect to or disconnect from a server. • Create and configure a WebLogic domain or domain template. • Exit WLST.
Deployment Commands	• Deploy, undeploy, and redeploy applications and standalone modules to a WebLogic Server instance. • Update an existing deployment plan. • Interrogate the WebLogic Deployment Manager object. • Start and stop a deployed application.
Diagnostics Commands	Export diagnostic data.
Editing Commands	Interrogate and edit configuration beans.
Information Commands	Interrogate domains, servers, and variables, and provide configuration bean, runtime bean, and WLST-related information.
Life Cycle Commands	Manage the life cycle of a server instance.
Node Manager Commands	Start, shut down, restart, and monitor WebLogic Server instances using Node Manager.
Tree Commands	Navigate among MBean hierarchies.

Browse Commands

Use the WLST browse commands, listed in Table A-2, to navigate the hierarchy of configuration or runtime beans and control the prompt display.

Table A-2 Browse Commands for WLST Configuration

Use this command...	To...	Use with WLST...
"cd" on page A-3	Navigate the hierarchy of configuration or runtime beans.	Online or Offline
"currentTree" on page A-4	Return the current location in the hierarchy.	Online
"prompt" on page A-5	Toggle the display of path information at the prompt.	Online or Offline
"pwd" on page A-6	Display the current location in the hierarchy.	Online or Offline

cd

Command Category: Browse Commands
Use with WLST: Online or Offline

Description

Navigates the hierarchy of configuration or runtime beans. This command uses a model that is similar to navigating a file system in a Windows or UNIX command shell. For example, to navigate back to a parent configuration or runtime bean, enter `cd('..')`. The character string .. (dot-dot), refers to the directory immediately above the current directory. To get back to the root bean after navigating to a bean that is deep in the hierarchy, enter `cd('/')`.

You can navigate to beans in the current hierarchy and to any child or instance.

The `cd` command returns a stub of the configuration or runtime bean instance, if one exists. If you navigate to a type, this command returns a stub of the configuration or runtime bean instance from which you navigated. In the event of an error, the command returns a `WLSTException`.

Note: The `cmo` variable is initialized to the root of all domain configuration beans when you first connect WLST to a server instance. It reflects the parent configuration bean type until you navigate to an instance. For more information about the `cmo` variable, see "Changing the Current Management Object" on page 4-3.

Syntax

```
cd(mbeanName)
```

Argument	Definition
mbeanName	Path to the bean in the namespace.

Examples

The following example navigates the hierarchy of configuration beans. The first command navigates to the `Servers` configuration bean type, the second, to the `myserver` configuration bean instance, and the last back up two levels to the original directory location.

```
wls:/mydomain/serverConfig> cd('Servers')
wls:/mydomain/serverConfig/Servers> cd('myserver')
wls:/mydomain/serverConfig/Servers/myserver> cd('../..')
wls:/mydomain/serverConfig>
```

currentTree

Command Category: Browse Commands
Use with WLST: Online

Description

Returns the current location in the hierarchy. This command enables you to store the current location in the hierarchy and easily return to it after browsing. In the event of an error, the command returns a `WLSTException`.

Syntax

```
currentTree()
```

Example

The following example stores the current location in the hierarchy in `myTree` and uses it to navigate back to the Edit MBean hierarchy from the runtime MBean hierarchy on an Administration Server instance.

```
wls:/mydomain/edit> myTree=currentTree()
wls:/mydomain/edit> serverRuntime()
Location changed to serverRuntime tree. This is a read-only tree with
ServerRuntimeMBean as the root.
For more help, use help('serverRuntime')

wls:/mydomain/serverRuntime> myTree()
wls:/mydomain/edit>
```

prompt

Command Category: Browse Commands
Use with WLST: Online or Offline

Description

Toggles the display of path information at the prompt, when entered without an argument. This command is useful when the prompt becomes too long due to the length of the path.

You can also explicitly specify `on` or `off` as an argument to the command. When you specify `off`, WLST hides the WLST prompt and defaults to the Jython prompt. By default, the WLST prompt displays the configuration or runtime navigation path information.

When you disable the prompt details, to determine your current location in the hierarchy, you can use the `pwd` command, as described in "pwd" on page A-6.

In the event of an error, the command returns a `WLSTException`.

Syntax

`prompt(`*`myPrompt`*`)`

Argument	Definition
myPrompt	Optional. Hides or displays WLST prompt. Valid values include `off` or `on`. • The `off` argument hides the WLST prompt. If you run `prompt('off')`, when using WLST online, the prompt defaults to the Jython prompt. You can create a new prompt using Jython syntax. For more information about programming using Jython, see `http://www.jython.org`. In this case, if you subsequently enter the `prompt` command without arguments, WLST displays the WLST command prompt without the path information. To redisplay the path information, enter `prompt()` again, or enter `prompt('on')`. • The `on` argument displays the default WLST prompt, including the path information.

Examples

The following example hides and then redisplays the path information at the prompt.

```
wls:/mydomain/serverConfig/Servers/myserver> prompt()
wls:/> prompt()
wls:/mydomain/serverConfig/Servers/myserver>
```

The following example hides the prompt and defaults to the Jython prompt (since the command is run using WLST online), changes the Jython prompt, and then redisplays the WLST prompt. This example also demonstrates the use of the `pwd` command.

Note: For more information about programming using Jython, see `http://www.jython.org`.

```
wls:/mydomain/serverConfig/Servers/myserver> prompt('off')
>>>sys.ps1="myprompt>"
myprompt> prompt()
wls:> pwd()
'serverConfig:Servers/myserver'
wls:> prompt()
wls:/mydomain/serverConfig/Servers/myserver>
```

pwd

Command Category: Browse Commands
Use with WLST: Online or Offline

Description

Displays the current location in the configuration or runtime bean hierarchy.

This command is useful when you have turned off the prompt display of the path information using the `prompt` command, as described in "prompt" on page A-5.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
pwd()
```

Example

The following example displays the current location in the configuration bean hierarchy.

```
wls:/mydomain/serverConfig/Servers/myserver/Log/myserver> pwd()
'serverConfig:/Servers/myserver/Log/myserver'
```

Control Commands

Use the WLST control commands, listed in Table A-3, to perform the following tasks:

- Connect to or disconnect from a server
- Create and configure a WebLogic domain or domain template, similar to the Configuration Wizard
- Exit WLST

Table A-3 lists the control commands for WLST configuration.

Table A-3 Control Commands for WLST Configuration

In order to...	Use this command...	To...	Use with WLST...
Connect to and disconnect from a WebLogic Server instance	"connect" on page A-10	Connect WLST to a WebLogic Server instance.	Online or Offline
	"disconnect" on page A-15	Disconnect WLST from a WebLogic Server instance.	Online

Table A-3 Control Commands for WLST Configuration (Continued)

In order to...	Use this command...	To...	Use with WLST...
Create a new domain from a domain template	"createDomain" on page A-14	Create a new domain using the specified template.	Offline
	"readTemplate" on page A-18	Open an existing domain template for domain creation.	Offline
	"writeDomain" on page A-19	Write the domain configuration information to the specified directory.	Offline
	"closeTemplate" on page A-10	Close the current domain template.	Offline
Update an existing domain (offline)	"readDomain" on page A-17	Open an existing domain for updating.	Offline
	"addTemplate" on page A-8	Extend the current domain using an application or service extension template.	Offline
	"updateDomain" on page A-19	Update and save the current domain.	Offline
	"closeDomain" on page A-9	Close the current domain.	Offline
Write a domain template	"writeTemplate" on page A-21	Writes the configuration information to the specified domain template file.	Offline
Exit WLST	"exit" on page A-15	Exit WLST from the interactive session and close the scripting shell.	Online or Offline

addTemplate

Command Category: Control Commands
Use with WLST: Offline

Description

Extends the current domain using an application or service extension template. In the event of an error, the command returns a `WLSTException`.

Syntax

`addTemplate(`*`templateFileName`*`)`

Argument	Definition
templateFileName	Name of the application or service extension template.

Example

The following example opens a domain and extends it using the specified extension template, `DefaultWebApp.jar`.

```
wls:/offline> readDomain('c:/bea/user_projects/domains/wlw')
wls:/offline/wlw> addTemplate('c:/bea/weblogic91/common/templates/
applications/DefaultWebApp.jar')
wls:/offline/wlw>
```

closeDomain

Command Category: Control Commands
Use with WLST: Offline

Description

Closes the current domain. The domain is no longer available for editing once it is closed. In the event of an error, the command returns a `WLSTException`.

Syntax

`closeDomain()`

Example

The following example closes the current domain:

```
wls:/offline> readDomain('c:/bea/user_projects/domains/medrec')
...
```

```
wls:/offline/medrec> updateDomain()
wls:/offline/medrec> closeDomain()
wls:/offline>
```

closeTemplate

Command Category: Control Commands
Use with WLST: Offline

Description

Closes the current domain template. The domain template is no longer available once it is closed. In the event of an error, the command returns a `WLSTException`.

Syntax

```
closeTemplate()
```

Example

The following example opens an existing domain template, performs some operations, and then closes the current domain template.

```
wls:/offline> readTemplate('c:/bea/weblogic81/common/templates/domains/
wls.jar')
...
wls:/offline/wls> closeTemplate()
wls:/offline>
```

connect

Command Category: Control Commands
Use with WLST: Online or Offline

Description

Connects WLST to a WebLogic Server instance.

You can specify the username and password on the command line, or you can specify an encrypted password that is stored locally by specifying the locations of the user configuration and key files as arguments to the `connect` command. For information about creating the user configuration and key files, see "storeUserConfig" on page A-90.

If you run the `connect` command without specifying the username and password, WLST attempts to process the command using one of the methods listed below (in order of precedence):

1. WLST searches for the default user configuration and key files that contain an encrypted username and password. This information must be valid for your current domain.
2. If the `connect` command was run from the domain directory in which the server was started, WLST attempts to load the username and password from the `boot.properties` file.
3. WLST prompts for a username, password, and URL.

If you do not specify the Administration Server name, the argument defaults to `AdminServer`.

Please note:

- BEA Systems strongly recommends that you connect WLST to the server through the SSL port or administration port. If you do not, the following warning message is displayed:

  ```
  Warning: An insecure protocol was used to connect to the server. To ensure
  on-the-wire security, the SSL port or Admin port should be used instead.
  ```

- If you are connecting to a WebLogic Server instance through an SSL listen port on a server that is using the demonstration SSL keys and certificates, you should invoke WLST using the following command:

  ```
  java -Dweblogic.security.SSL.ignoreHostnameVerification=true
  -Dweblogic.security.TrustKeyStore=DemoTrust weblogic.WLST
  ```

  For more information about invoking WLST, see "Main Steps for Using WLST" on page 2-8.

- If you are connecting to a WebLogic Server instance via HTTP, ensure that the `TunnelingEnabled` attribute is set to `true` for the WebLogic Server instance. For more information, see "TunnelingEnabled" in *WebLogic Server MBean Reference*.

After successfully connecting to a WebLogic Server instance, all the local variables are initialized.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
connect([username, password], [url], [adminServerName])

connect([userConfigFile, userKeyFile], [adminServerName])
```

Argument	Definition
username	Optional. Username of the operator who is connecting WLST to the server. If not specified, WLST processes the command as described above.
password	Optional. Password of the operator who is connecting WLST to the server. If not specified, WLST processes the command as described above.
url	Optional. Listen address and listen port of the server instance, specified using the following format: `[`*protocol*`://]`*listen-address*`:`*listen-port*. If not specified, this argument defaults to `t3://localhost:7001`.
userConfigFile	Optional. Name and location of a user configuration file which contains an encrypted username and password. When you create a user configuration file, the `storeUserConfig` command uses a key file to encrypt the username and password. Only the key file that encrypts a user configuration file can decrypt the username and password. (See "storeUserConfig" on page A-90.)
userKeyFile	Optional. Name and location of the key file that is associated with the specified user configuration file and is used to decrypt it. (See "storeUserConfig" on page A-90.)
adminServerName	Optional. Name of the Administration Server. This value is used when WLST is invoked from a domain directory. This argument defaults to `AdminServer`.

Examples

The following example connects WLST to a WebLogic Server instance. In this example, the Administration Server name defaults to `AdminServer`. Note that a warning is displayed if the SSL or administration port is not used to connect to the server.

```
wls:/offline> connect('weblogic','weblogic','t3://localhost:8001')
Connecting to weblogic server instance running at t3://localhost:8001 as
username weblogic...

Successfully connected to Admin Server 'AdminServer' that belongs to domain
'mydomain'.

Warning: An insecure protocol was used to connect to the server. To ensure
on-the-wire security, the SSL port or Admin port should be used instead.
```

```
wls:/mydomain/serverConfig>
```

The following example connects WLST to a WebLogic Server instance at the specified URL. In this example, the username and password are passed as variables. This example uses a secure protocol.

```
wls:/offline> username = 'weblogic'
wls:/offline> password = 'weblogic'
wls:/offline> connect(username,password,'t3s://myhost:8001')
Connecting to weblogic server instance running at t3://myhost:8001 as
username weblogic...

Successfully connected to Admin Server 'AdminServer' that belongs to domain
'mydomain'.
wls:/mydomain/serverConfig>
```

The following example connects WLST to a WebLogic Server instance using a user configuration and key file to provide user credentials. The Administration Server name defaults to `AdminServer`.

```
wls:/offline> connect(userConfigFile='c:/myfiles/myuserconfigfile.secure',
userKeyFile='c:/myfiles/myuserkeyfile.secure')
Connecting to weblogic server instance running at t3://localhost:7001 as
username ...

Successfully connected to Admin Server 'AdminServer' that belongs to domain
'mydomain'.
wls:/mydomain/serverConfig>
```

createDomain

Command Category: Control Commands
Use with WLST: Offline

Description

Creates a domain using the specified template.

Note: If you wish to modify the domain configuration settings when creating a domain, see Option 2 in "Creating a Domain (Offline)" on page 3-2.

The `createDomain` command is similar in functionality to the `unpack` command, as described in *Creating Templates and Domains Using the pack and unpack Commands*.

In the event of an error, the command returns a `WLSTException`.

Syntax

`createDomain(`*`domainTemplate`*`, `*`domainDir`*`, `*`user`*`, `*`password`*`)`

Argument	Definition
domainTemplate	Name and location of the domain template from which you want to create a domain.
domainDir	Name of the directory to which you want to write the domain configuration information.
user	Name of the default user.
password	Password of the default user.

Example

The following example creates a new domain using the Avitek MedRec template and sets the default username and password to `weblogic`. The domain is saved to the following directory: `c:/bea/user_projects/domains/medrec`.

```
wls:/offline> createDomain('c:/bea/weblogic91/common/templates/domains
/wls_medrec.jar','c:/bea/user_projects/domains/medrec', 'weblogic',
'weblogic')
```

disconnect

Command Category: Control Commands
Use with WLST: Online

Description

Disconnects WLST from a WebLogic Server instance. The `disconnect` command does not cause WLST to exit the interactive scripting shell; it closes the current WebLogic Server instance connection and resets all the variables while keeping the interactive shell alive.

In the event of an error, the command returns a `WLSTException`.

You can connect to another WebLogic Server instance using the `connect` command, as described in "connect" on page A-10.

Syntax

`disconnect(`*`force`*`)`

Argument	Definition
force	Optional. Boolean value specifying whether WLST should disconnect without waiting for the active sessions to complete. This argument defaults to `false`, indicating that all active sessions must complete before disconnect.

Example

The following example disconnects from a running server:

```
wls:/mydomain/serverConfig> disconnect()
Disconnected from weblogic server: myserver
wls:/offline>
```

exit

Command Category: Control Commands
Use with WLST: Online or Offline

Description

Exits WLST from the user session and closes the scripting shell.

If there is an edit session in progress, WLST prompts you for confirmation. To skip the prompt, set the *defaultAnswer* argument to `y`.

By default, WLST calls `System.exit(0)` for the current WLST JVM when exiting WLST. If you would like the JVM to exit with a different exit code, you can specify a value using the *exitCode* argument.

Note: When the WLST exit command is issued within an Ant script, it may also exit the execution of the Ant script. It is recommended that when invoking WLST within an Ant script, you fork a new JVM by specifying `fork="true"`.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
exit([defaultAnswer], [exitcode])
```

Argument	Definition
defaultAnswer	Optional. Default response, if you would prefer not to be prompted at the command line. Valid values are `y` and `n`. This argument defaults to null, and WLST prompts you for a response.
exitcode	Optional. Exit code to set when exiting WLST.

Example

The following example disconnects from the user session and closes the scripting shell.

```
wls:/mydomain/serverConfig> exit()
Exiting WebLogic Scripting Tool ...
c:\>
```

The following example disconnects from the user session, closes the scripting shell, and sets the error code to 101.

```
wls:/mydomain/serverConfig> exit(exitcode=101)
Exiting WebLogic Scripting Tool ...
c:\>
```

readDomain

Command Category: Control Commands
Use with WLST: Offline

Description

Opens an existing domain for updating.

When you open a template or domain, WLST is placed into the configuration bean hierarchy for that domain, and the prompt is updated to reflect the current location in the configuration hierarchy. For example:

```
wls:/offline/base_domain>
```

WebLogic Server configuration beans exist within a hierarchical structure. In the WLST file system, the hierarchies correspond to drives; types and instances are directories; attributes and operations are files. WLST traverses the hierarchical structure of configuration beans using commands such as `cd`, `ls`, and `pwd` in a similar way that you would navigate a file system in a UNIX or Windows command shell. After navigating to an configuration bean instance, you interact with the bean using WLST commands. For more information, see "Navigating and Interrogating MBeans" on page 4-1.

In the event of an error, the command returns a `WLSTException`.

Syntax

`readDomain(`*`domainDirName`*`)`

Argument	Definition
domainDirName	Name of the domain directory that you wish to open.

Example

The following example opens the `medrec` domain for editing.

```
wls:/offline> readDomain('c:/bea/user_projects/domains/medrec')
wls:/offline/medrec>
```

readTemplate

Command Category: Control Commands
Use with WLST: Offline

Description

Opens an existing domain template for domain creation.

When you open a domain template, WLST is placed into the configuration bean hierarchy for that domain template, and the prompt is updated to reflect the current location in the configuration hierarchy. For example:

```
wls:/offline/base_domain>
```

WebLogic Server configuration beans exist within a hierarchical structure. In the WLST file system, the hierarchies correspond to drives; types and instances are directories; attributes and operations are files. WLST traverses the hierarchical structure of configuration beans using commands such as `cd`, `ls`, and `pwd` in a similar way that you would navigate a file system in a UNIX or Windows command shell. After navigating to a configuration bean instance, you interact with the bean using WLST commands. For more information, see "Navigating and Interrogating MBeans" on page 4-1.

Note: Using WLST and a domain template, you can only create and access security information when you are creating a new domain. When you are updating a domain, you cannot access security information through WLST.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
readTemplate(templateFileName)
```

Argument	Definition
templateFileName	Name of the JAR file corresponding to the domain template.

Example

The following example opens the `medrec.jar` domain template for domain creation.

```
wls:/offline> readTemplate('c:/bea/weblogic91/common/templates/domains
/wls_medrec.jar')
wls:/offline/wls_medrec>
```

updateDomain

Command Category: Control Commands
Use with WLST: Offline

Description

Updates and saves the current domain. The domain continues to be editable after you update and save it.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
updateDomain()
```

Example

The following examples opens the medrec domain, performs some operations, and updates and saves the current domain:

```
wls:/offline> readDomain('c:/bea/user_projects/domains/medrec')
...
wls:/offline/medrec> updateDomain()
```

writeDomain

Command Category: Control Commands
Use with WLST: Offline

Description

Writes the domain configuration information to the specified directory.

Once you write the domain to file system, you can continue to update the domain template object that exists in memory, and reissue the `writeDomain` command to store the domain configuration to a new or existing file.

By default, when you write a domain, the associated applications are written to *BEAHOME*/user_projects/applications/*domainname*, where *BEAHOME* specifies the BEA home directory and *domainname* specifies the name of the domain. This directory must be empty; otherwise, WLST displays an error.

When you have finished using the domain template object in memory, close it using the closeTemplate command. If you want to edit the domain that has been saved to disk, you can open it using the readDomain command.

Note: The name of the domain is derived from the name of the domain directory. For example, for a domain saved to c:/bea/user_projects/domains/myMedrec, the domain name is myMedrec.

Before writing the domain, you must define a password for the default user, if it is not already defined. For example:

```
cd('/Security/base_domain/User/weblogic')
cmo.setPassword('weblogic')
```

In the event of an error, the command returns a WLSTException.

Syntax

```
writeDomain(domainDir)
```

Argument	Definition
domainDir	Name of the directory to which you want to write the domain configuration information.

Example

The following example reads the medrec.jar domain templates, performs some operations, and writes the domain configuration information to the c:/bea/user_projects/domains/medrec directory.

```
wls:/offline> readTemplate('c:/bea/weblogic81/common/templates/domains
/wls.jar')
...
wls:/offline/base_domain>
writeDomain('c:/bea/user_projects/domains/base_domain')
```

writeTemplate

Command Category: Control Commands
Use with WLST: Offline

Description

Writes the domain configuration information to the specified domain template. You can use the domain configuration template to recreate the domain.

Once your write the configuration information to the domain configuration template, you can continue to update the domain or domain template object that exists in memory, and reissue the `writeDomain` or `writeTemplate` command to store the domain configuration to a new or existing domain or domain template file. For more information, see "writeDomain" on page A-19 or "writeTemplate" on page A-21, respectively.

In the event of an error, the command returns a `WLSTException`.

Note: The `writeTemplate` command is similar in functionality to the `pack` command, as described in *Creating Templates and Domains Using the pack and unpack Commands*. However, `writeTemplate` does not support creating a Managed Server template.

Syntax

`writeTemplate(`*`templateName`*`)`

Argument	Definition
templateName	Name of the domain template to store the domain configuration information.

Example

The following example writes the current domain configuration to the domain template named `c:/bea/user_projects/templates/myTemplate.jar`.

```
wls:/offline> readDomain('c:/bea/user_projects/domains/mydomain')
...
wls:/offline/base_domain>
writeTemplate('c:/bea/user_projects/templates/myTemplate.jar')
```

Deployment Commands

Use the WLST deployment commands, listed in Table A-4, to:

- Deploy, undeploy, and redeploy applications and standalone modules to a WebLogic Server instance.
- Update an existing deployment plan.
- Interrogate the WebLogic Deployment Manager object.
- Start and stop a deployed application.

For more information about deploying applications, see *Deploying Applications to WebLogic Server*.

Table A-4 Deployment Commands for WLST Configuration

This command...	Enables you to...	Use with WLST...
"deploy" on page A-23	Deploy an application to a WebLogic Server instance.	Online
"distributeApplication" on page A-27	Copy the deployment bundle to the specified targets.	Online
"getWLDM" on page A-28	Return the WebLogic `DeploymentManager` object.	Online
"loadApplication" on page A-29	Load an application and deployment plan into memory.	Online
"redeploy" on page A-30	Redeploy a previously deployed application	Online
"startApplication" on page A-31	Start an application, making it available to users.	Online
"stopApplication" on page A-33	Stop an application, making it unavailable to users.	Online
"undeploy" on page A-34	Undeploy an application from the specified servers.	Online
"updateApplication" on page A-35	Updates an application configuration using a new deployment plan.	Online

deploy

Command Category: Deployment Commands
Use with WLST: Online

Description

Deploys an application to a WebLogic Server instance.

The `deploy` command returns a `WLSTProgress` object that you can access to check the status of the command. For more information about the `WLSTProgress` object, see "WLSTProgress Object" on page C-4. In the event of an error, the command returns a `WLSTException`.

Note: If there is an edit session in progress, the `deploy` command does not block user interaction.

Syntax

```
deploy(appName, path, [targets], [stageMode], [planPath], [options])
```

Argument	Definition
appName	Name of the application or standalone J2EE module to be deployed.
path	Name of the application directory, archive file, or root of the exploded archive directory to be deployed.
targets	Optional. Comma-separated list of the target. Each target may be qualified with a J2EE module name (for example, *module1@server1*) enabling you to deploy different modules of the application archive on different servers. This argument defaults to the server to which WLST is currently connected.
stageMode	Optional. Staging mode for the application you are deploying. Valid values are `stage`, `nostage`, and `external_stage`. For information about the staging modes, see "Controlling Deployment File Copying with Staging Modes" in *Deploying Applications to WebLogic Server*. This argument defaults to null.
planPath	Optional. Name of the deployment plan file. The filename can be absolute or relative to the application directory. This argument defaults to the `plan/plan.xml` file in the application directory, if one exists.

Argument	Definition (Continued)
options	Optional. Comma-separated list of deployment options, specified as name-value pairs. Valid options include: • **altDD**—Location of the alternate application deployment descriptor on the Administration Server. • **altWlsDD**—Location of the alternate WebLogic application deployment descriptor on the Administration Server. • **archiveVersion**—Archive version number. • **block**—Boolean value specifying whether WLST should block user interaction until the command completes. This option defaults to `true`. If set to `false`, WLST returns control to the user after issuing the command; you can query the `WLSTProgress` object to determine the status of the command. If you are importing WLST as a Jython module, as described in "Importing WLST as a Jython Module" on page 2-11, *`block`* is always set to `true`. • **clusterDeploymentTimeout**—Time, in milliseconds, granted for a cluster deployment task on this application. • **createPlan**—Boolean value indicating that user would like to create a default plan. This option defaults to `false`.

Argument	Definition (Continued)
options (Continued)	• **defaultSubmoduleTargets**—Boolean value indicating that targeting for any JMS submodules should be derived by the system. • **forceUndeployTimeout**—Force undeployment timeout value. • **gracefulIgnoreSessions**—Boolean value specifying whether the graceful production to admin mode operation should ignore pending HTTP sessions. This option defaults to `false` and only applies if `gracefulProductionToAdmin` is set to `true`. • **gracefulProductionToAdmin**—Boolean value specifying whether the production to Admin mode operation should be graceful. This option defaults to `false`. • **libImplVersion**—Implementation version of the library, if it is not present in the manifest. • **libraryModule**—Boolean value specifying whether the module is a library module. This option defaults to `false`. • **libSpecVersion**—Specification version of the library, if it is not present in the manifest. • **planVersion**—Plan version number. • **retireGracefully**—Retirement policy to gracefully retire an application only after it has completed all in-flight work. This policy is only meaningful for stop and redeploy operations and is mutually exclusive to the retire timeout policy. • **retireTimeout**—Time (in seconds) WLST waits before retiring an application that has been replaced with a newer version. This option default to `-1`, which specifies graceful timeout. • **securityModel**—Security model. Valid values include: `DDOnly`, `CustomRoles`, `CustomRolesAndPolicy`, and `Advanced`. • **securityValidationEnabled**—Boolean value specifying whether security validation is enabled. • **subModuleTargets**—Submodule level targets for JMS modules. For example, `submod@mod-jms.xml@target | submoduleName@target`. • **testMode**—Boolean value specifying whether to start the Web application with restricted access. This option defaults to `false`. • **timeout**—Time (in milliseconds) that WLST waits for the deployment process to complete before canceling the operation. A value of 0 indicates that the operation will not time out. This argument defaults to 300,000 ms (or 5 minutes). • **versionIdentifier**—Version identifier.

Example

The following example deploys the businessApp application located at c:/myapps/business, A default deployment plan is created.

The deploy command returns a WLSTProgress object that you can access to check the status of the command. The WLSTProgress object is captured in a user-defined variable, in this case, progress.

```
wls:/mydomain/serverConfig/Servers> progress=
deploy(appName='businessApp',path='c:/myapps/business',createplan='true')
```

The previous example stores the WLSTProgress object returned in a user-defined variable, in this case, progress. You can then use the progress variable to print the status of the deploy command. For example:

```
wls:/mydomain/serverConfig/Servers> progress.printStatus()
Current Status of your Deployment:
Deployment command type: deploy
Deployment State       : completed
Deployment Message     : null
wls:/mydomain/serverConfig/Servers>
```

For more information about the WLSTProgress object, see "WLSTProgress Object" on page C-4.

The following example deploys the demoApp application in the archive file located at c:/myapps/demos/app/demoApp.ear, targeting the application modules to myserver, and using the deployment plan file located in c:/myapps/demos/app/plan/plan.xml. WLST waits 120,000 ms for the process to complete.

```
wls:/mydomain/serverConfig/Servers> deploy('demoApp',
'c:/myapps/demos/app/demoApp.ear', targets='myserver',
planPath='c:/myapps/demos/app/plan/plan.xml', timeout=120000)
```

The following example deploys the jmsApp application located at c:/myapps/demos/jmsApp/demo-jms.xml, targeting the application module to a specific target.

```
wls:/mydomain/serverConfig/Servers>deploy('jmsApp',path='c:/myapps/demos/j
msApps/demo-jms.xml', subModuleTargets='jmsApp@managed1')
```

The following example shows how to set the application version (appVersion) to a unique identifier to support production (side-by-side) redeployment. This example deploys the demoApp

application in the archive file located at `c:/myapps/demos/app/demoApp.ear`, and sets the application and archive version numbers to the specified values.

```
wls:/mydomain/serverConfig> deploy('demoApp',
'c:/myapps/demos/app/demoApp.ear', archiveVersion='901-101',
appVersion='901-102')
```

For more information about production redeployment strategies, see "Updating Applications in a Production Environment" in *Deploying Applications to WebLogic Server*.

distributeApplication

Command Category: Deployment Commands
Use with WLST: Online

Description

Copies the deployment bundle to the specified targets. The deployment bundle includes module, configuration data, and any additional generated code. The `distributeApplication` command does not start deployment.

The `distributeApplication` command returns a `WLSTProgress` object that you can access to check the status of the command. For more information about the `WLSTProgress` object, see "WLSTProgress Object" on page C-4. In the event of an error, the command returns a `WLSTException`.

Syntax

```
distributeApplication(appPath, [planPath], [targets], [options])
```

Argument	Definition
appPath	Name of the archive file or root of the exploded archive directory to be deployed.
planPath	Optional. Name of the deployment plan file. The filename can be absolute or relative to the application directory. This argument defaults to the `plan/plan.xml` file in the application directory, if one exists.
targets	Optional. Comma-separated list of targets. Each target may be qualified with a J2EE module name (for example, *module1@server1*) enabling you to deploy different modules of the application archive on different servers. This argument defaults to the server to which WLST is currently connected.

Argument	Definition (Continued)
options	Optional. Comma-separated list of deployment options, specified as name-value pairs. For a list of valid deployment options, see the *options* argument description in "deploy" on page A-23.

Example

The following example loads the BigApp application located in the c:/myapps directory, and stores the WLSTProgress object in a user-defined variable, in this case, progress.

The following example distributes the c:/myapps/BigApp application to the myserver, oamserver1, and oamcluster servers, using the deployment plan defined at c:/deployment/BigApp/plan.xml.

```
wls:/offline> progress=distributeApplication('c:/myapps/BigApp',
'c:/deployment/BigApp/plan.xml', 'myserver,oamserver1,oamcluster')
Distributing Application and Plan ...
Successfully distributed the application.
```

The previous example stores the WLSTProgress object in a user-defined variable, in this case, progress. You can then use the progress variable to determine if the distributeApplication command has completed. For example:

```
wls:/mydomain/serverConfig/Servers> progress.isCompleted()
1
wls:/mydomain/serverConfig/Servers>
```

For more information about the WLSTProgress object, see "WLSTProgress Object" on page C-4.

getWLDM

Command Category: Deployment Commands
Use with WLST: Online

Description

Returns the WebLogic DeploymentManager object. You can use the object methods to configure and deploy applications. WLST must be connected to an Administration Server to run this command. In the event of an error, the command returns a WLSTException.

Syntax

```
getWLDM()
```

Example

The following example gets the `WebLogicDeploymentManager` object and stores it in the `wldm` variable.

```
wls:/mydomain/serverConfig> wldm=getWLDM()
wls:/mydomain/serverConfig> wldm.isConnected()
1
wls:/mydomain/serverConfig>
```

loadApplication

Command Category: Deployment Commands
Use with WLST: Online

Description

Loads an application and deployment plan into memory.

The `loadApplication` command returns a `WLSTPlan` object that you can access to make changes to the deployment plan. For more information about the `WLSTPlan` object, see "WLSTPlan Object" on page C-1. In the event of an error, the command returns a `WLSTException`.

Syntax

```
loadApplication(appPath, [planPath], [createPlan])
```

Argument	Definition
appPath	Name of the top-level parent application directory, archive file, or root of the exploded archive directory containing the application to be loaded.
planPath	Optional. Name of the deployment plan file. The filename can be absolute or relative to the application directory. This argument defaults to the `plan/plan.xml` file in the application directory, if one exists.
createPlan	Optional. Boolean value specifying whether WLST should create a plan in the application directory if the specified plan does not exist. This argument defaults to `true`.

Example

The following example loads the `c:/myapps/myejb.jar` application using the plan file at `c:/myplans/myejb/plan.xml`.

```
wls:/myserver/serverConfig> myPlan=loadApplication('c:/myapps/myejb.jar',
'c:/myplans/myejb/plan.xml')
Loading application from c:/myapps/myejb.jar and deployment plan from
c:/myplans/myejb/plan.xml ...
Successfully loaded the application.
wls:/myserver/serverConfig>
```

The previous example stores the `WLSTPlan` object returned in the `myPlan` variable. You can then use `myPlan` variable to display information about the plan, such as the variables. For example:

```
wls:/myserver/serverConfig> myPlan.showVariables()
MyEJB jndi.ejb
MyWAR app.foo
wls:/myserver/serverConfig>
```

For more information about the `WLSTPlan` object, see "WLSTPlan Object" on page C-1.

redeploy

Command Category: Deployment Commands
Use with WLST: Online

Description

Reloads classes and redeploys a previously deployed application.

The `redeploy` command returns a `WLSTProgress` object that you can access to check the status of the command. For more information about the `WLSTProgress` object, see "WLSTProgress Object" on page C-4.

In the event of an error, the command returns a `WLSTException`.

For more information about redeploying applications, see "Overview of Common Deployment Scenarios" in *Deploying Application to WebLogic Server*.

Syntax

```
redeploy(appName, [planPath], [options])
```

Argument	Definition
appName	Name of the application to be redeployed.
planPath	Optional. Name of the deployment plan file. The filename can be absolute or relative to the application directory. This argument defaults to the `plan/plan.xml` file in the application directory, if one exists.
options	Optional. Comma-separated list of deployment options, specified as name-value pairs. For a list of valid deployment options, see *options* argument description in "deploy" on page A-23. In addition, the following deployment option can be specified for the `redeploy` command: • **appPath**—Name of the archive file or root of the exploded archive directory to be redeployed..

Example

The following example redeploys `myApp` application using the `plan.xml` file located in the `c:/myapps` directory.

```
wls:/mydomain/serverConfig> progress=redeploy('myApp'
'c:/myapps/plan.xml')
Redeploying application 'myApp' ...
Redeployment of 'myApp' is successful
wls:/mydomain/serverConfig>
```

The previous example stores the `WLSTProgress` object returned in a user-defined variable, in this case, `progress`. You can then use the `progress` variable to access the state of the `redeploy` command. For example:

```
wls:/mydomain/serverConfig/Servers> progress.getState()
'completed'
wls:/mydomain/serverConfig/Servers>
```

For more information about the `WLSTProgress` object, see "WLSTProgress Object" on page C-4.

startApplication

Command Category: Deployment Commands
Use with WLST: Online

Description

Starts an application, making it available to users. The application must be fully configured and available in the domain.

The `startApplication` command returns a `WLSTProgress` object that you can access to check the status of the command. For more information about the `WLSTProgress` object, see "WLSTProgress Object" on page C-4. In the event of an error, the command returns a `WLSTException`.

Syntax

```
startApplication(appName, [options])
```

Argument	Definition
appName	Name of the application to start, as specified in the `plan.xml` file.
options	Optional. Comma-separated list of deployment options, specified as name-value pairs. For a list of valid deployment options, see *options* argument description in "deploy" on page A-23.

Example

The following example starts the `BigApp` application with the specified deployment options.

```
wls:/offline> progress=startApplication('BigApp', stageMode='NOSTAGE',
testMode='false')
Starting the application...
Successfully started the application.
```

The previous example stores the `WLSTProgress` object returned in a user-defined variable, in this case, `progress`. You can then use the `progress` variable to access the state of the `startApplication` command. For example:

```
wls:/mydomain/serverConfig/Servers> progress.getState()
'completed'
wls:/mydomain/serverConfig/Servers>
```

For more information about the `WLSTProgress` object, see "WLSTProgress Object" on page C-4.

stopApplication

Command Category: Deployment Commands
Use with WLST: Online

Description

Stops an application, making it unavailable to users. The application must be fully configured and available in the domain.

The `stopApplication` command returns a `WLSTProgress` object that you can access to check the status of the command. For more information about the `WLSTProgress` object, see "WLSTProgress Object" on page C-4.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
stopApplication(appName, [options])
```

Argument	Definition
appName	Name of the application to stop, as specified in the `plan.xml` file.
options	Optional. Comma-separated list of deployment options, specified as name-value pairs. For a list of valid deployment options, see *options* argument description in "deploy" on page A-23.

Example

The following example stops the `BigApp` application.

```
wls:/offline> progress=stopApplication('BigApp')
Stopping the application...
Successfully stopped the application.
```

The previous example stores the `WLSTProgress` object returned in a user-defined variable, in this case, `progress`. You can then use the `progress` variable to check whether `stopApplication` command is running. For example:

```
wls:/mydomain/serverConfig/Servers> progress.isRunning()
0
wls:/mydomain/serverConfig/Servers>
```

For more information about the `WLSTProgress` object, see "WLSTProgress Object" on page C-4.

undeploy

Command Category: Deployment Commands
Use with WLST: Online

Description

Undeploys an application from the specified servers.

The `undeploy` command returns a `WLSTProgress` object that you can access to check the status of the command. For more information about the `WLSTProgress` object, see "WLSTProgress Object" on page C-4. In the event of an error, the command returns a `WLSTException`.

For more information about deploying and undeploying applications, see "Overview of Common Deployment Scenarios" in *Deploying Applications to WebLogic Server*.

Syntax

```
undeploy(appName,[targets],[options])
```

Argument	Definition
appName	Deployment name for the deployed application.
targets	Optional. List of the target servers from which the application will be removed. If not specified, defaults to all current targets.
options	Optional. Comma-separated list of deployment options, specified as name-value pairs. For a list of valid deployment options, see *options* argument description in "deploy" on page A-23.

Example

The following example removes the `businessApp` application from all target servers. WLST waits 60,000 ms for the process to complete.

```
wls:/mydomain/serverConfig> undeploy('businessApp', timeout=60000)
Undeploying application businessApp ...
<Jul 20, 2005 9:34:15 AM EDT> <Info> <J2EE Deployment SPI> <BEA-260121>
```

```
<Initiating undeploy operation for application, businessApp [archive:
null],
to AdminServer .>

Completed the undeployment of Application with status
Current Status of your Deployment:
Deployment command type: undeploy
Deployment State       : completed
Deployment Message     : no message
wls:/mydomain/serverConfig>
```

updateApplication

Command Category: Deployment Commands
Use with WLST: Online

Description

Updates an application configuration using a new deployment plan. The application must be fully configured and available in the domain.

The `updateApplication` command returns a `WLSTProgress` object that you can access to check the status of the command. For more information about the `WLSTProgress` object, see "WLSTProgress Object" on page C-4. In the event of an error, the command returns a `WLSTException`.

Syntax

```
updateApplication(appName, [planPath], [options])
```

Argument	Definition
appName	Name of the application, as specified in the current `plan.xml` file.
planPath	Optional. Name of the new deployment plan file. The filename can be absolute or relative to the application directory.
options	Optional. Comma-separated list of deployment options, specified as name-value pairs. For a list of valid deployment options, see *options* argument description in "deploy" on page A-23.

Example

The following example updates the application configuration for `BigApp` using the `plan.xml` file located in `c:/myapps/BigApp/newPlan`.

```
wls:/offline> progress=updateApplication('BigApp',
'c:/myapps/BigApp/newPlan/plan.xml', stageMode='STAGE', testMode='false')
Updating the application...
Successfully updated the application.
```

The previous example stores the `WLSTProgress` object returned in a user-defined variable, in this case, `progress`. You can then use the `progress` variable to access the state of the `updateApplication` command. For example:

```
wls:/mydomain/serverConfig/Servers> progress.getState()
'completed'
wls:/mydomain/serverConfig/Servers>
```

For more information about the `WLSTProgress` object, see "WLSTProgress Object" on page C-4.

Diagnostics Commands

Use the WLST diagnostics commands, listed in Table A-5, to execute queries against the diagnostic data. For more information about the WebLogic Server Diagnostic Service, see *Configuring and Using the WebLogic Diagnostic Framework*.

Table A-5 Diagnostic Command for WLST Configuration

This command...	Enables you to...	Use with WLST...
"exportDiagnosticData" on page A-36	Execute a query against the specified log file.	Offline
"exportDiagnosticDataFromServer" on page A-38	Executes a query on the server side and retrieves the exported WebLogic Diagnostic Framework (WLDF) data.	Online

exportDiagnosticData

Command Category: Diagnostics Commands
Use with WLST: Offline

Description

Executes a query against the specified log file. The results are saved to an XML file.

For more information about the WebLogic Server Diagnostic Service, see *Configuring and Using the WebLogic Diagnostic Framework*.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
exportDiagnosticData([options])
```

Argument	Definition
options	Optional. Comma-separated list of export diagnostic options, specified as name-value pairs. Valid options include: • **beginTimestamp**—Timestamp (inclusive) of the earliest record to be added to the result set. This option defaults to 0. • **endTimestamp**—Timestamp (exclusive) of the latest record to be added to the result set. This option defaults to `Long.MAX_VALUE`. • **exportFileName**—Name of the file to which the data is exported. This option defaults to `export.xml`. • **logicalName**—Logical name of the log file being read. Valid values include: `HarvestedDataArchive`, `EventsDataArchive`, `ServerLog`, `DomainLog`, `HTTPAccessLog`, `WebAppLog`, `ConnectorLog`, and `JMSMessageLog`. This option defaults to `ServerLog`. • **logName**—Base log filename containing the log data to be exported. This option defaults to `myserver.log`. • **logRotationDir**—Directory containing the rotated log files. This option defaults to "`.`" (current directory). • **query**—Expression specifying the filter condition for the data records to be included in the result set. This option defaults to "" (empty string), which returns all data. For more information, see "WLDF Query Language" in *Configuring and Using the Weblogic Diagnostic Framework*. • **storeDir**—Location of the diagnostic store for the server. This option defaults to `../data/store/diagnostics`.

Example

The following example executes a query against the `ServerLog` named `myserver.log` and stores the results in the file named `myExport.xml`.

```
wls:/offline/mydomain>exportDiagnosticData(logicalName='ServerLog',
logName='myserver.log', exportFileName='myExport.xml')
{'elfFields': '', 'logName': 'myserver.log', 'logRotationDir': '.',
'endTimestamp': 9223372036854775807L, 'exportFileName': 'export.xml',
'storeDir': '../data/store/diagnostics', 'logicalName': 'ServerLog',
'query': '', 'beginTimestamp': 0}

Exporting diagnostic data to export.xml

<Aug 2, 2005 6:58:21 PM EDT> <Info> <Store> <BEA-280050> <Persistent store
"WLS_DIAGNOSTICS" opened:
directory="c:\bea\weblogic91\server\data\store\diagnostics"
writePolicy="Disabled" blockSize=512 directIO=false driver="wlfileio2">

wls:/mydomain/serverRuntime>
```

exportDiagnosticDataFromServer

Command Category: Diagnostics Commands
Use with WLST: Online

Description

Executes a query on the server side and retrieves the exported WebLogic Diagnostic Framework (WLDF) data. The results are saved to an XML file.

For more information about the WebLogic Server Diagnostic Service, see *Configuring and Using the WebLogic Diagnostic Framework*.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
exportDiagnosticDataFromServer([options])
```

Argument	Definition
options	Optional. Comma-separated list of export diagnostic options, specified as name-value pairs. Valid options include: • **beginTimestamp**—Timestamp (inclusive) of the earliest record to be added to the result set. This option defaults to 0. • **endTimestamp**—Timestamp (exclusive) of the latest record to be added to the result set. This option defaults to `Long.MAX_VALUE`. • **exportFileName**—Name of the file to which the data is exported. This option defaults to `export.xml`. • **logicalName**—Logical name of the log file being read. Valid values include: `HarvestedDataArchive`, `EventsDataArchive`, `ServerLog`, `DomainLog`, `HTTPAccessLog`, `WebAppLog`, `ConnectorLog`, and `JMSMessageLog`. This option defaults to `ServerLog`. • **query**—Expression specifying the filter condition for the data records to be included in the result set. This option defaults to "" (empty string), which returns all data.

Example

The following example executes a query against the `HTTPAccessLog` and stores the results in the file named `myExport.xml`.

```
wls:/mydomain/serverRuntime>
exportDiagnosticDataFromServer(logicalName="HTTPAccessLog",
exportFileName="myExport.xml")
```

Editing Commands

Use the WLST editing commands, listed in Table A-6, to interrogate and edit configuration beans.

Note: To edit configuration beans, you must be connected to an Administration Server, and you must navigate to the edit tree and start an edit session, as described in "edit" on page A-123 and "startEdit" on page A-62, respectively.

If you connect to a Managed Server, WLST functionality is limited to browsing the configuration bean hierarchy. While you cannot use WLST to change the values of MBeans on Managed Servers, it is possible to use the Management APIs to do so. BEA Systems recommends that you change only the values of configuration MBeans on the

Administration Server. Changing the values of MBeans on Managed Servers can lead to an inconsistent domain configuration.

For more information about editing configuration beans, see "Editing Configuration MBeans" on page 4-12.

Table A-6 Editing Commands for WLST Configuration

This command...	Enables you to...	Use with WLST...
"activate" on page A-41	Activate changes saved during the current editing session but not yet deployed.	Online or Offline
"assign" on page A-42	Assign resources to one or more destinations.	Offline
"assignAll" on page A-45	Assign all applications or services to one or more destinations.	Offline
"cancelEdit" on page A-46	Cancel an edit session, release the edit lock, and discard all unsaved changes. This operation can be called by any user with administrator privileges, even if the user did not start the edit session.	Online
"create" on page A-47	Create a configuration bean of the specified type for the current bean.	Online or Offline
"delete" on page A-49	Delete an instance of a configuration for the current configuration bean.	Online or Offline
"encrypt" on page A-50	Encrypt the specified string.	Online
"get" on page A-51	Return the value of the specified attribute.	Online or Offline
"getActivationTask" on page A-52	Return the latest `ActivationTask` MBean on which a user can get status.	Online
"invoke" on page A-53	Invokes a management operation on the current configuration bean.	Online
"isRestartRequired" on page A-54	Determine whether a server restart is required.	Online
"loadDB" on page A-55	Load SQL files into a database.	Offline

Table A-6 Editing Commands for WLST Configuration (Continued)

This command...	Enables you to...	Use with WLST...
"loadProperties" on page A-56	Load property values from a file.	Online or Offline
"save" on page A-56	Save the edits that have been made but have not yet been saved.	Online
"set" on page A-57	Set the specified attribute value for the current configuration bean.	Online or Offline
"setOption" on page A-58	Set options related to a domain creation or update.	Offline
"showChanges" on page A-61	Show the changes made to the configuration by the current user during the current edit session.	Online
"startEdit" on page A-62	Starts a configuration edit session on behalf of the currently connected user.	Online
"stopEdit" on page A-63	Stop the current edit session, release the edit lock, and discard unsaved changes.	Online
"unassign" on page A-64	Unassign applications or resources from one or more destinations.	Offline
"unassignAll" on page A-66	Unassign applications or resources from one or more destinations.	Offline
"undo" on page A-67	Revert all unsaved or unactivated edits.	Online
"validate" on page A-68	Validate the changes that have been made but have not yet been saved.	Online

activate

Command Category: Editing Commands
Use with WLST: Online

Description

Activates changes saved during the current editing session but not yet deployed. This command prints a message if a server restart is required for the changes that are being activated.

The `activate` command returns the latest `ActivationTask` MBean which reflects the state of changes that a user is currently making or has made recently. You can then invoke methods to get information about the latest Configuration Manager activate task in progress or just completed. In the event of an error, the command returns a `WLSTException`.

Syntax

```
activate([timeout], [block])
```

Argument	Definition
timeout	Optional. Time (in milliseconds) that WLST waits for the activation of configuration changes to complete before canceling the operation. A value of -1 indicates that the operation will not time out. This argument defaults to 300,000 ms (or 5 minutes).
block	Optional. Boolean value specifying whether WLST should block user interaction until the command completes. This argument defaults to `false`, indicating that user interaction is not blocked. In this case, WLST returns control to the user after issuing the command and assigns the task MBean associated with the current task to a variable that you can use to check its status.If you are importing WLST as a Jython module, as described in "Importing WLST as a Jython Module" on page 2-11, *block* is always set to `true`.

Example

The following example activates the changes made during the current edit session that have been saved to disk, but that have not yet been activated. WLST waits for 100,000 ms for the activation to complete, and 200,000 ms before the activation is stopped.

```
wls:/mydomain/edit !> activate(200000, block='true')
Activating all your changes, this may take a while ...
The edit lock associated with this edit session is released once the
activation is completed.
Action completed.
wls:/mydomain/edit>
```

assign

Command Category: Editing Commands
Use with WLST: Offline

Description

Assigns resources to one or more destinations.

In the event of an error, the command returns a `WLSTException`.

Syntax

`assign(`*`sourceType, sourceName, destinationType, destinationName`*`)`

Argument	Definition
`sourceType`	Type of configuration bean to be assigned. This value can be set to one of the following values: • `AppDeployment` • `Library` • *`securityType`* (such as `User`) • `Server` • *`service`* (such as `JDBCSystemResource`) • *`service`*`.SubDeployment`, where *`service`* specifies the service type of the `SubDeployment` (such as `JMSSystemResource.SubDeployment`); you can also specify nested subdeployments (such as `AppDeployment.SubDeployment.SubDeployment`) Guidelines for setting this value are provided below.
`sourceName`	Name of the resource to be assigned. Multiple names can be specified, separated by commas, or you can use the wildcard (*) character to specify all resources of the specified type. Specify subdeployments using the following format: *`service.subDeployment`*, where *`service`* specifies the parent service and *`subDeployment`* specifies the name of the subdeployment. For example, `myJMSResource.myQueueSubDeployment.` You can also specify nested subdeployments, such as `MedRecEAR.MedRecAppScopedJMS.MedRecJMSServer`.
`destinationType`	Type of destination. Guidelines for setting this value are provided below.
`destinationName`	Name of the destination. Multiple names can be specified, separated by commas.

Use the following guidelines for setting the `sourceType` and `destinationType`:

- When assigning **application deployments**, set the values as follows:
  - *sourceType*: AppDeployment
  - *destinationType*: Target
- When assigning **libraries**, set the values as follows:
  - *sourceType*: Library
  - *destinationType*: Target
- When assigning **services**, set the values as follows:
  - *sourceType*: Name of the specific server, such as JDBCSystemResource
  - *destinationType*: Target
- When assigning **servers** to **clusters**, set the values as follows:
  - *sourceType*: Server
  - *destinationType*: Cluster
- When assigning **subdeployments**, set the values as follows:
  - *sourceType*: *service*.SubDeployment, where *service* specifies the parent of the SubDeployment, such as JMSSystemResource.SubDeployment; you can also specify nested subdeployments (such as AppDeployment.SubDeployment.SubDeployment)
  - *destinationType*: Target
- When assigning **security types**, set the values as follows:
  - *sourceType*: Name of the security type, such as User
  - *destinationType*: Name of the destination security type, such as Group

Example

The following examples:

- Assign the servers myServer and myServer2 to the cluster myCluster.

```
wls:/offline/mydomain> assign("Server", "myServer,myServer2", "Cluster",
"myCluster")
```

- Assign all servers to the cluster myCluster.

```
wls:/offline/mydomain> assign("Server", "*", "Cluster", "myCluster")
```

- Assign the application deployment `myAppDeployment` to the target server `newServer`.

```
wls:/offline/mydomain> assign("AppDeployment", "myAppDeployment",
"Target", "newServer")
```

- Assign the user `newUser` to the group `Monitors`.

```
wls:/offline/mydomain> assign("User", "newUser", "Group", "Monitors")
```

- Assign the SubDeployment `myQueueSubDeployment`, which is a child of the JMS resource `myJMSResource`, to the target server `newServer`.

```
wls:/offline/mydomain> assign('JMSSystemResource.SubDeployment',
'myJMSResource.myQueueSubDeployment', 'Target', 'newServer')
```

- Assign the nested SubDeployment `MedRecAppScopedJMS.MedRecJMSServer`, which is a child of the AppDeployment `AppDeployment`, to the target server `AdminServer`.

```
wls:/offline/mydomain>assign('AppDeployment.SubDeployment.SubDeployment
','MedRecEAR.MedRecAppScopedJMS.MedRecJMSServer','Target','AdminServer'
)
```

assignAll

Command Category: Editing Commands
Use with WLST: Offline

Description

Note: This command is deprecated as of WebLogic Server 9.0. You should update your script to use the `assign` command as described in "assign" on page A-42. This command will still operate on any resources that exist for the specified *`sourceType`*.

Assigns all applications or services to one or more destinations.

Note: Note that you must assign JMS server and JMS distributed destinations using the `assign` command, as described in "assign" on page A-42.

In the event of an error, the command returns a `WLSTException`.

Syntax

`assignAll(`*`sourceType, destinationType, destinationName`*`)`

Argument	Definition
sourceType	Type of applications or services to be assigned. This value can be set to `Applications` or `Services`.
destinationType	Type of destination. This value must be set to `Target`.
destinationName	Name(s) of the destination. Multiple names can be specified, separated by commas.

Example

The following example assigns all services to the servers `adminServer` and `cluster1`.

```
wls:/offline/mydomain> assignAll("Services", "Target",
"adminServer,cluster1")
```

The following services, if present, are assigned to the specified targets: `MigratableRMIService`, `Shutdownclass`, `Startupclass`, `FileT3`, `RMCFactory`, `MailSession`, `MessagingBridge`, `JMSConnectionFactory`, `JDBCConnectionPool`, `JDBCMultipool`, `JDBCTxDatasource`, `JDBCDataSource`, `JDBCPoolComp`, `JoltConnectionPool`, `WLECConnectionPool`, and `WTCServer`.

cancelEdit

Command Category: Editing Commands
Use with WLST: Online

Description

Cancels an edit session, releases the edit lock, and discards all unsaved changes.

The user issuing this command does not have to be the current editor; this allows an administrator to cancel an edit session, if necessary, to enable other users to start an edit session.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
cancelEdit([defaultAnswer])
```

Argument	Definition
defaultAnswer	Optional. Default response, if you would prefer not to be prompted at the command line. Valid values are `y` and `n`. This argument defaults to null, and WLST prompts you for a response.

Example

The following example cancels the current editing session. WLST prompts for verification before canceling.

```
wls:/mydomain/edit !> cancelEdit()
Sure you would like to cancel the edit session? (y/n)y
Edit session is cancelled successfully
wls:/mydomain/edit>
```

create

Command Category: Editing Commands
Use with WLST: Online or Offline

Description

Creates a configuration bean of the specified type for the current bean.

The `create` command returns a stub for the newly created configuration bean. In the event of an error, the command returns a `WLSTException`.

Notes: Child types must be created under an instance of their parent type. You can only create configuration beans that are children of the current Configuration Management Object (`cmo`) type. For more information about the `cmo` variable, see "Changing the Current Management Object" on page 4-3.

Because WLST offline enables you to access and update the configuration objects that appear in the configuration files only, if you wish to view and/or change attribute values for a configuration object that is not already persisted in the configuration files as an XML element, you must first create the configuration object.

Please note the following when using the `create` command with **WLST online**:

- You must be connected to an Administration Server. You cannot use the `create` command for runtime MBeans or when WLST is connected to a Managed Server instance.
- You must navigate to the edit configuration MBean hierarchy using the `edit` command before issuing this command. See "edit" on page A-123.
- You can use the create command to create a WebLogic Server configuration MBean that is a child of the current MBean type.

Please note the following when using the `create` command with **WLST offline**:

- When using WLST offline, the following characters are not valid in object names: period (`.`), forward slash (`/`), or backward slash (`\`).

For more information about:

- Creating MBeans, see "Understanding WebLogic Server MBeans" in *Developing Custom Management Utilities with JMX*.
- Examples of creating specific types of MBean resources, for example, a JMS or JDBC system resource, refer to the WLST sample scripts installed with your product, as described in "WLST Sample Scripts" on page 1-3.
- MBeans, their child types, attributes, and operations, see *WebLogic Server MBean Reference*.

Syntax

```
create(name, childMBeanType, [baseProviderType])
```

Argument	Definition
name	Name of the configuration bean that you are creating.
childMBeanType	Type of configuration bean that you are creating. You can create instances of any type defined in the `config.xml` file except custom security types. For more information about valid configuration beans, see *WebLogic Server MBean Reference*.
baseProviderType	When creating a security provider, specifies the base security provider type, for example, `AuthenticationProvider`. This argument defaults to None.

Example

The following example creates a child configuration bean of type `Server` named `newServer` for the current configuration bean, storing the stub as `server1`:

```
wls:/mydomain/edit !> server1=create('newServer','Server')
Server with name 'newServer' has been created successfully.
wls:/mydomain/edit !> server1.getName()
'newServer'
wls:/mydomain/edit !>
```

The following example creates an authentication provider security provider called `myProvider`:

```
wls:/mydomain/edit !> cd('SecurityConfiguration/mydomain/Realms/myrealm')
wls:/mydomain/edit !>
create('myProvider','weblogic.security.providers.authentication.SQLAuthent
icator','AuthenticationProvider')
```

The following example creates a machine named `highsec_nm` and sets attributes for the associated Node Manager.

```
wls:/mydomain/edit !> create('highsec_nm', 'Machine')
wls:/mydomain/edit !> cd('Machine/highsec_nm/NodeManager/highsec_nm')
wls:/mydomain/edit !> set('DebugEnabled', 'true')
wls:/mydomain/edit !> set('ListenAddress', 'innes')
wls:/mydomain/edit !> set('NMType', 'SSL')
wls:/mydomain/edit !> set('ShellCommand', '')
```

delete

Command Category: Editing Commands
Use with WLST: Online or Offline

Description

Deletes an instance of a configuration bean of the specified type for the current configuration bean.

In the event of an error, the command returns a `WLSTException`.

Note: You can only delete configuration beans that are children of current Configuration Management Object (`cmo`) type. For more information about the `cmo` variable, see "Changing the Current Management Object" on page 4-3.

Syntax

```
delete(name, childMBeanType)
```

Argument	Definition
name	Name of the child configuration bean to delete.
childMBeanType	Type of the configuration bean to be deleted. You can delete instances of any type defined in the `config.xml` file. For more information about valid configuration beans, see *WebLogic Server MBean Reference*.

Example

The following example deletes the configuration bean of type `Server` named `newServer`:

```
wls:/mydomain/edit !> delete('newServer','Server')
Server with name 'newServer' has been deleted successfully.
wls:/mydomain/edit !>
```

encrypt

Command Category: Editing Commands
Use with WLST: Online

Description

Encrypts the specified string. You can then use the encrypted string in your configuration file or as an argument to a command.

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

Note: An encrypted string must have been encrypted by the encryption service in the WebLogic Server domain where it will be used. If not, the server will not be able to decrypt the string.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
encrypt(obj, [domainDir])
```

Argument	Definition
obj	String that you want to encrypt.
domainDir	Optional. Name of the domain directory containing the `security/SerializedSystemIni.dat` file to use to encrypt the file. This argument defaults to the directory from which WLST was invoked.

Example

The following example encrypts the specified string using the `security/SerializedSystemIni.dat` file in the specified domain directory.

```
wls:/mydomain/serverConfig>
es=encrypt('myPassword','c:/bea/domains/mydomain')
```

get

Command Category: Editing Commands
Use with WLST: Online or Offline

Description

Returns the value of the specified attribute. For more information about the MBean attributes that can be viewed, see *WebLogic Server MBean Reference*. In the event of an error, the command returns a `WLSTException`.

Note: You can list all attributes and their current values by entering `ls('a')`. For more information, see "ls" on page A-81.

Alternatively, you can use the `cmo` variable to perform any get method on the current configuration bean. For example:

```
cmo.getListenPort()
```

For more information about the `cmo` variable, see "Changing the Current Management Object" on page 4-3.

Syntax

```
get(attrName)
```

Argument	Definition
attrName	Name of the attribute to be displayed. You can specify the full pathname of the attribute. If no pathname is specified, the attribute is displayed for the current configuration object.

Example

The following example returns the value of the `AdministrationPort` for the current configuration bean.

```
wls:/mydomain/serverConfig> get('AdministrationPort')
9002
```

Alternatively, you can use the `cmo` variable:

```
cmo.getAdministrationPort()
```

getActivationTask

Command Category: Editing Commands
Use with WLST: Online

Description

Return the latest `ActivationTask` MBean on which a user can get status. The `ActivationTask` MBean reflects the state of changes that a user is currently making or has made recently. You can then invoke methods to get information about the latest Configuration Manager activate task in progress or just completed. In the event of an error, the command returns a `WLSTException`.

Syntax

```
getActivationTask()
```

Example

The following example returns the latest `ActivationTask` MBean on which a user can get status and stores it within the task variable.

```
wls:/mydomain/serverConfig> task=getActivationTask()
wls:/mydomain/serverConfig> task.getState()
STATE_COMMITTED
```

invoke

Command Category: Editing Commands
Use with WLST: Online

Description

Invokes a management operation on the current configuration bean. Typically, you use this command to invoke operations other than the `get` and `set` operations that most WebLogic Server configuration beans provide. The class objects are loaded through the same class loader that is used for loading the configuration bean on which the action is invoked.

You cannot use the `invoke` command when WLST is connected to a Managed Server instance.

If successful, the `invoke` command returns the object that is returned by the operation invoked. In the event of an error, the command returns a `WLSTException`.

Syntax

`invoke(`*`methodName`*`, `*`parameters`*`, `*`signatures`*`)`

Argument	Definition
methodName	Name of the method to be invoked.
parameters	An array of parameters to be passed to the method call.
signatures	An array containing the signature of the action.

Example

The following example invokes the `lookupServer` method on the current configuration bean.

```
wls:/mydomain/config> objs =
jarray.array([java.lang.String("oamserver")],java.lang.Object)
wls:/mydomain/edit> strs = jarray.array(["java.lang.String"],java.lang.String)
wls:/mydomain/edit> invoke('lookupServer',objs,strs)
true
wls:/mydomain/edit>
```

isRestartRequired

Command Category: Editing Commands
Use with WLST: Online

Description

Determines whether a server restart is required.

If you invoke this command while an edit session is in progress, the response is based on the edits that are currently in progress. If you specify the name of an attribute, WLST indicates whether a server restart is required for that attribute only.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
isRestartRequired([attributeName])
```

Argument	Definition
attributeName	Optional. Name of a specific attribute for which you want to check if a server restart is required.

Example

The following example specifies whether a server restart is required for all changes made during the current WLST session.

```
wls:/mydomain/edit !> isRestartRequired()
Server re-start is REQUIRED for the set of changes in progress.

The following attribute(s) have been changed on MBeans that require server
re-start.
MBean Changed : mydomain:Name=mydomain,Type=Domain
Attributes changed : AutoConfigurationSaveEnabled
```

The following example specifies whether a server restart is required if you edit the `ConsoleEnabled` attribute.

```
wls:/mydomain/edit !> isRestartRequired("ConsoleEnabled")
Server re-start is REQUIRED if you change the attribute ConsoleEnabled
wls:/mydomain/edit !>
```

loadDB

Command Category: Editing Commands
Use with WLST: Offline

Description

Loads SQL files into a database.

The `loadDB` command loads the SQL files from a template file. This command can only be issued after a domain template or extension template has been loaded into memory using the `readTemplate` or `addTemplate` command, respectively.

Before executing this command, ensure that the following conditions are true:

- The appropriate database is running.
- SQL files exist for the specified database and version.

  To verify that the appropriate SQL files exist, open the domain template and locate the relevant SQL file list, `jdbc.index`, in the `_jdbc_` directory. For example, for PointBase version 4.4, the SQL file list is located at `_jdbc_\Pointbase\44\jdbc.index`.

The command fails if the above conditions are not met.

In the event of an error, the command returns a `WLSTException`.

Syntax

`loadDB(`*`dbVersion, connectionPoolName`*`)`

Argument	Definition
`dbVersion`	Version of the database for which the SQL files are intended to be used.
`connectionPoolName`	Name of the JDBC connection pool to be used to load SQL files.

Example

The following example loads SQL files, intended for version 4.4 of the database, using the `myPool-PointBase` JDBC connection pool:

```
wls:/offline/mydomain> loadDB('4.4', 'myPool-PointBase')
```

loadProperties

Command Category: Information Commands
Use with WLST: Online and Offline

Description

Loads property values from a file and makes them available in the WLST session.

This command cannot be used when you are importing WLST as a Jython module, as described in "Importing WLST as a Jython Module" on page 2-11.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
loadProperties(fileName)
```

Argument	Definition
fileName	Properties file pathname.

Example

This example gets and sets the properties file values.

```
wls:/mydomain/serverConfig> loadProperties('c:/temp/myLoad.properties')
```

save

Command Category: Editing Commands
Use with WLST: Online

Description

Saves the edits that have been made but have not yet been saved. This command is only valid when an edit session is in progress. For information about starting an edit session, see "startEdit" on page A-62.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
save()
```

Example

The following example saves the edits that have not yet been saved to disk.

```
wls:/mydomain/edit !> save()
Saving all your changes ...
Saved all your changes successfully.
wls:/mydomain/edit !>
```

set

Command Category: Editing Commands
Use with WLST: Online or Offline

Description

Sets the specified attribute value for the current configuration bean. For more information about the MBean attributes that can be set, see *WebLogic Server MBean Reference*.

In the event of an error, the command returns a `WLSTException`.

Note: You can list all attributes and their current values by entering `ls('a')`. For more information, see "ls" on page A-81.

When you use this command for a domain configuration MBean, the new values are saved in the `config.xml` file.

Please note the following when using **WLST online**:

- You cannot use the `set` command when WLST is connected to a Managed Server instance.
- While you cannot use WLST to change the values of MBeans on Managed Servers, it is possible to use the Management APIs to do so. BEA Systems recommends that you change only the values of configuration MBeans on the Administration Server. Changing the values of MBeans on Managed Servers can lead to an inconsistent domain configuration.

Alternatively, you can use the `cmo` variable to perform any `set` method on the current configuration bean. For example:

```
cmo.setPassword(attributeValue)
```

For more information about the cmo variable, see "Changing the Current Management Object" on page 4-3.

Syntax

```
set(attrName, value)
```

Argument	Definition
attrName	Name of the attribute to be set.
value	Value of the attribute to be set. **Note:** This value should not be enclosed in single or double quotes.

Example

The following example sets the ArchiveConfigurationCount attribute of DomainMBean to 10:

```
wls:/mydomain/serverConfig> set('ArchiveConfigurationCount',10)
```

setOption

Command Category: Editing Commands
Use with WLST: Offline

Description

Sets options related to a domain creation or update. In the event of an error, the command returns a WLSTException.

Syntax

```
setOption(optionName, optionValue)
```

Argument	Definition
optionName	Name of the option to set. Available options for **domain creation** include: • CreateStartMenu—Boolean value specifying whether to create a Start Menu shortcut on a Windows platform. This option defaults to true. **Note:** If a user with Administrator privileges installed the software and chose to create the Start menu entries in the All Users folder, only users with Administrator privileges can create Start menu entries in the same folder when creating a domain using the Configuration Wizard or WLST. That is, if a user without Administrator privileges uses the Configuration Wizard or WLST from this installation to create domains, Start menu shortcuts to the domains are not created. In this case, the users can manually create shortcuts in their local Start menu folder, if desired. • DomainName—Name of the domain. By default, the name of the domain is derived from the name of the domain directory. For example, for a domain saved to c:/bea/user_projects/domains/myMedrec, the domain name is myMedrec. By setting DomainName, the name of the created domain will be independent of the domain directory name. • JavaHome—Home directory for the JVM to be used when starting the server. This option defaults to the Sun JDK installed with the product. • OverwriteDomain—Boolean value specifying whether to allow an existing domain to be overwritten. This option defaults to false. • ServerStartMode—Mode to use when starting the server for the newly created domain. This value can be dev (development) or prod (production). This option defaults to dev. Available options for **domain updates** include: • AllowCasualUpdate—Boolean value specifying whether to allow a domain to be updated without adding an extension template. This option defaults to true. • ReplaceDuplicates—Boolean value specifying whether to keep original configuration elements in the domain or replace the elements with corresponding ones from an extension template when there is a conflict. This option defaults to true. Available options for both **domain creation** and **domain updates** include: • AppDir—Application directory to be used when a separate directory is desired for applications, as specified by the template. This option defaults to *BEAHOME*/user_projects/applications/*domainname*, where *BEAHOME* specifies the BEA home directory and *domainname* specifies the name of the domain. • AutoAdjustSubDeploymentTarget—Boolean value specifying whether WLST automatically adjusts targets for the subdeployments of AppDeployments. This option defaults to true. To deactivate this feature, set the option to false and explicitly set the targeting for AppDeployment subdeployments before writing or updating the domain or domain template. • AutoDeploy—Boolean value specifying whether to activate auto deployment when a cluster or multiple Managed Servers are created. This option defaults to true. To deactivate this feature, set the option to false on the first line of your script.

Argument	Definition (Continued)
optionValue	Value for the option. **Note:** Boolean values can be specified as a String (`true`, `false`) or integer (`0`, `1`).

Example

The following example sets the `CreateStartMenu` option to `false`:

```
wls:/offline> setOption('CreateStartMenu', 'false')
```

showChanges

Command Category: Editing Commands
Use with WLST: Online

Description

Shows the changes made to the configuration by the current user during the current edit session. In the event of an error, the command returns a `WLSTException`.

Syntax

```
showChanges([onlyInMemory])
```

Argument	Definition
onlyInMemory	Optional. Boolean value specifying whether to display only the changes that have not yet been saved. This argument defaults to `false`, indicating that all changes that have been made from the start of the session are displayed.

Example

The following example shows all of the changes made by the current user to the configuration since the start of the current edit session.

```
wls:/mydomain/edit !> showChanges()
Changes that are in memory and saved to disc but not yet activated are:

MBean Changed           : com.bea:Name=basicWLSDomain,Type=Domain
```

```
Operation Invoked       : add
Attribute Modified      : Machines
Attributes Old Value    : null
Attributes New Value    : Mach1
Server Restart Required : false

MBean Changed           : com.bea:Name=basicWLSDomain,Type=Domain
Operation Invoked       : add
Attribute Modified      : Servers
Attributes Old Value    : null
Attributes New Value    : myserver
Server Restart Required : false
```

startEdit

Command Category: Editing Commands
Use with WLST: Online

Description

Starts a configuration edit session on behalf of the currently connected user. You must navigate to the edit configuration MBean hierarchy using the `edit` command before issuing this command. For more information, see "edit" on page A-123.

This command must be called prior to invoking any command to modify the domain configuration.

In the event of an error, the command returns a `WLSTException`.

Note: WLST automatically starts an edit session if it detects that there is an edit session that is already in progress by the same user, which may have been started via the Administration Console or another WLST session.

Syntax

```
startEdit([waitTimeInMillis], [timeoutInMillis], [exclusive])
```

Argument	Definition
waitTimeInMillis	Optional. Time (in milliseconds) that WLST waits until it gets a lock, in the event that another user has a lock. This argument defaults to 0 ms.

Argument	Definition (Continued)
timeoutInMillis	Optional. Timeout (in milliseconds) that WLST waits to release the edit lock. This argument defaults to -1 ms, indicating that this edit session never expires.
exclusive	Optional. Specifies whether the edit session should be an exclusive session. If set to `true`, if the same owner enters the `startEdit` command, WLST waits until the current edit session lock is released before starting the new edit session. The exclusive lock times out according to the time specified in *timeoutInMillis*. This argument defaults to `false`.

Example

The following example saves the edits that have not yet been saved to disk.

```
wls:/mydomain/edit> startEdit(60000, 120000)
Starting an edit session ...
Started edit session, please be sure to save and activate your changes once
you are done.
wls:/mydomain/edit !>
```

stopEdit

Command Category: Editing Commands
Use with WLST: Online

Description

Stops the current edit session, releases the edit lock, and discards unsaved changes.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
stopEdit([defaultAnswer])
```

Argument	Definition
defaultAnswer	Optional. Default response, if you would prefer not to be prompted at the command line. Valid values are `y` and `n`. This argument defaults to null, and WLST prompts you for a response.

Example

The following example stops the current editing session. WLST prompts for verification before canceling.

```
wls:/mydomain/edit !> stopEdit()
Sure you would like to stop your edit session? (y/n)
y
Edit session has been stopped successfully.
wls:/mydomain/edit>
```

unassign

Command Category: Editing Commands
Use with WLST: Offline

Description

Unassign applications or resources from one or more destinations.

In the event of an error, the command returns a `WLSTException`.

Syntax

`unassign(`*`sourceType, sourceName, destinationType, destinationName`*`)`

Argument	Definition
sourceType	Type of configuration bean to be unassigned. This value can be set to one of the following values: • `AppDeployment` • `Library` • *securityType* (such as `User`) • `Server` • *service* (such as `JDBCSystemResource`) • *service*`.SubDeployment`, where *service* specifies the service type of the `SubDeployment` (such as `JMSSystemResource.SubDeployment`); you can also specify nested subdeployments (such as `AppDeployment.SubDeployment.SubDeployment`)

Argument	Definition (Continued)
sourceName	Name of the application or resource to be unassigned. Multiple names can be specified, separated by commas, or you can use the wildcard (*) character to specify all resources of the specified type. Specify subdeployments using the following format: *service.subDeployment*, where *service* specifies the parent service and *subDeployment* specifies the name of the subdeployment. For example, `myJMSResource.myQueueSubDeployment.` You can also specify nested subdeployments, such as `MedRecEAR.MedRecAppScopedJMS.MedRecJMSServer`.
destinationType	Type of destination. Guidelines for setting this value are provided below.
destinationName	Name of the destination. Multiple names can be specified, separated by commas.

Use the following guidelines for setting the `sourceType` and `destinationType`:

- When unassigning **application deployments**, set the values as follows:
  - *sourceType*: `AppDeployment`
  - *destinationType*: `Target`
- When unassigning **libraries**, set the values as follows:
  - *sourceType*: `Library`
  - *destinationType*: `Target`
- When unassigning **security types**, set the values as follows:
  - *sourceType*: Name of the security type, such as `User`
  - *destinationType*: Name of the destination security type, such as `Group`
- When unassigning **servers** from **clusters**, set the values as follows:
  - *sourceType*: `Server`
  - *destinationType*: `Cluster`
- When unassigning **services**, set the values as follows:
  - *sourceType*: Name of the specific server, such as `JDBCSystemResource`
  - *destinationType*: `Target`

- When unassigning **subdeployments**, set the values as follows:
  - *sourceType*: *service*.SubDeployment, where *service* specifies the parent of the SubDeployment, such as JMSSystemResource.SubDeployment; you can also specify nested subdeployments (such as AppDeployment.SubDeployment.SubDeployment)
  - *destinationType*: Target

Example

The following examples:

- Unassign the servers myServer and myServer2 from the cluster myCluster.

```
wls:/offline/medrec> unassign("Server", "myServer,myServer2", "Cluster",
"myCluster")
```

- Unassign all servers from the cluster myCluster.

```
wls:/offline/mydomain> unassign("Server", "*", "Cluster", "myCluster")
```

- Unassign the user newUser from the group Monitors.

```
wls:/offline/medrec> unassign("User", "newUser", "Group", "Monitors")
```

- Unassign the application deployment myAppDeployment from the target server newServer.

```
wls:/offline/mydomain> unassign("AppDeployment", "myAppDeployment",
"Target", "newServer")
```

- Unassign the nested SubDeployment MedRecAppScopedJMS.MedRecJMSServer, which is a child of the AppDeployment AppDeployment, from the target server AdminServer.

```
wls:/offline/mydomain>
assign('AppDeployment.SubDeployment.SubDeployment',
'MedRecEAR.MedRecAppScopedJMS.MedRecJMSServer', 'Target','AdminServer')
```

unassignAll

Command Category: Editing Commands
Use with WLST: Offline

Description

Note: This command is deprecated as of WebLogic Server 9.0. You should update your script to use the unassign command as described in "unassign" on page A-64. This command will still operate on any resources that exist for the specified *sourceType*.

Unassigns all applications or services from one or more destinations.

In the event of an error, the command returns a `WLSTException`.

Syntax

`unassignAll(`*`sourceType, destinationType, destinationName`*`)`

Argument	Definition
`sourceType`	Type of applications or services to be unassigned. This value can be set to `Applications` or `Services`.
`destinationType`	Type of destination. This value must be set to `Target`.
`destinationName`	Name(s) of the destination. Multiple names can be specified, separated by commas.

Example

The following example unassigns all services from the servers `adminServer` and `cluster1`.

```
wls:/offline/medrec> unassignAll("Services", "Target",
"adminServer,cluster1")
```

The following services, if present, are unassigned from the specified targets: `MigratableRMIService`, `Shutdownclass`, `Startupclass`, `FileT3`, `RMCFactory`, `MailSession`, `MessagingBridge`, `JMSConnectionFactory`, `JDBCConnectionPool`, `JDBCMultipool`, `JDBCTxDatasource`, `JDBCDataSource`, `JDBCPoolComp`, `JoltConnectionPool`, `WLECConnectionPool`, and `WTCServer`.

undo

Command Category: Editing Commands
Use with WLST: Online

Description

Reverts all unsaved or unactivated edits.

You specify whether to revert all unactivated edits (including those that have been saved to disk), or all edits made since the last `save` operation. This command does not release the edit session.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
undo([unactivateChanges], [defaultAnswer])
```

Argument	Definition
unactivateChanges	Optional. Boolean value specifying whether to undo all unactivated changes, including edits that have been saved to disk. This argument defaults to `false`, indicating that all edits since the last `save` operation are reverted.
defaultAnswer	Optional. Default response, if you would prefer not to be prompted at the command line. Valid values are `y` and `n`. This argument defaults to null, and WLST prompts you for a response.

Example

The following example reverts all changes since the last `save` operation. WLST prompts for verification before reverting.

```
wls:/mydomain/edit !> undo()
Sure you would like to undo your changes? (y/n)
y
Discarded your in-memory changes successfully.
wls:/mydomain/edit>
```

The following example reverts all unactivated changes. WLST prompts for verification before reverting.

```
wls:/mydomain/edit !> undo('true')
Sure you would like to undo your changes? (y/n)
y
Discarded all your changes successfully.
wls:/mydomain/edit>
```

validate

Command Category: Editing Commands
Use with WLST: Online

Description

Validates the changes that have been made but have not yet been saved. This command enables you to verify that all changes are valid before saving them.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
validate()
```

Example

The following example validates all changes that have been made but have not yet been saved.

```
wls:/mydomain/edit !> validate()
Validating changes ...
Validated the changes successfully
```

Information Commands

Use the WLST information commands, listed in Table A-7, to interrogate domains, servers, and variables, and provide configuration bean, runtime bean, and WLST-related information.

Table A-7 Information Commands for WLST Configuration

This command...	Enables you to...	Use with WLST...
"addListener" on page A-71	Add a JMX listener to the specified MBean.	Online
"configToScript" on page A-72	Convert an existing server configuration (`config` directory) to an executable WLST script	Online or Offline
"dumpStack" on page A-74	Display stack trace from the last exception that occurred while performing a WLST action, and reset the stack trace.	Online or Offline
"dumpVariables" on page A-74	Display all variables used by WLST, including their name and value.	Online or Offline
"find" on page A-75	Find MBeans and attributes in the current hierarchy.	Online

Table A-7 Information Commands for WLST Configuration (Continued)

This command...	Enables you to...	Use with WLST...
"getConfigManager" on page A-77	Return the latest `ConfigurationManagerBean` MBean which manages the change process.	Online
"getMBean" on page A-77	Return the MBean by browsing to the specified path.	Online
"getMBI" on page A-78	Return the `MBeanInfo` for the specified `MBeanType` or the `cmo` variable.	Online
"getPath" on page A-79	Return the MBean path for the specified MBean instance.	Online
"listChildTypes" on page A-79	List all the children MBeans that can be created or deleted for the `cmo` type.	Online
"lookup" on page A-80	Look up the specified MBean.	Online
"ls" on page A-81	List all child beans and/or attributes for the current configuration or runtime bean.	Online or Offline
"man" on page A-84	Display help from `MBeanInfo` for the current MBean or its specified attribute.	Online
"redirect" on page A-85	Redirect WLST output to the specified filename.	Online or Offline
"removeListener" on page A-86	Remove a listener that was previously defined.	Online
"showListeners" on page A-87	Show all listeners that are currently defined.	Online
"startRecording" on page A-87	Record all user interactions with WLST; useful for capturing commands to replay.	Online or Offline
"state" on page A-88	Returns a map of servers or clusters and their state using Node Manager.	Online
"stopRecording" on page A-89	Stop recording WLST commands.	Online or Offline
"stopRedirect" on page A-90	Stop redirection of WLST output to a file.	Online or Offline

Table A-7 Information Commands for WLST Configuration (Continued)

This command...	Enables you to...	Use with WLST...
"storeUserConfig" on page A-90	Create a user configuration file and an associated key file.	Online
"threadDump" on page A-92	Display a thread dump for the specified server.	Online or Offline
"viewMBean" on page A-93	Display information about an MBean, such as the attribute names and values, and operations.	Online
"writeIniFile" on page A-94	Convert WLST definitions and method declarations to a Python (`.py`) file.	Online or Offline

addListener

Command Category: Information Commands
Use with WLST: Online

Description

Adds a JMX listener to the specified MBean. Any changes made to the MBean are reported to standard out and/or are saved to the specified configuration file.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
addListener(mbean, [attributeNames], [logFile], [listenerName])
```

Argument	Definition
mbean	Name of the MBean or MBean object to listen on.
attributeNames	Optional. Comma-separated list of all attribute names on which you would like to add a JMX listener. This argument defaults to null, and adds a JMX listener for all attributes.
logFile	Optional. Name and location of the log file to which you want to write listener information.This argument defaults to standard out.

Argument	Definition (Continued)
listenerName	Optional. Name of the JMX listener. This argument defaults to a WLST-generated name.

Example

The following example defines a JMX listener on the `cmo` MBean for the `Notes` and `ArchiveConfigurationCount` attributes. The listener is named `domain-listener` and is stored in `./listeners/domain.log`.

```
wls:/mydomain/serverConfig> addListener(cmo,
"Notes,ArchiveConfigurationCount","./listeners/domain.log","domain-listene
r")
```

configToScript

Command Category: Information Commands
Use with WLST: Online or Offline

Converts an existing server configuration (`config` directory) to an executable WLST script. You can use the resulting script to re-create the resources on other servers.

Before running the generated script, you should update the properties file to specify values that are appropriate for your environments. When you run the generated script:

- If a server is currently running, WLST will try to connect using the values in the properties file and then run the script commands to create the server resources.
- If no server is currently running, WLST will start a server with the values in the properties file, run the script commands to create the server resources, and shutdown the server. This may cause you to exit from the WLST shell.

The `configToScript` command creates a user configuration file and an associated key file to store encrypted attributes. The user configuration file contains the encrypted information. The key file contains a secret key that is used to encrypt and decrypt the encrypted information.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
configToScript([configPath], [pyPath], [overwrite], [propertiesFile],
[createDeploymentScript])
```

Argument	Definition
configPath	Optional. Path to the `config` directory that contains the configuration that you want to convert. This argument defaults to `./config`.
pyPath	Optional. Path and filename to which you want to write the converted WLST script. This argument defaults to `./config/config.py`.
overwrite	Optional. Boolean value specifying whether the script file should be overwritten if it already exists. This argument defaults to `true`, indicating that the script file is overwritten.
propertiesFile	Optional. Path to the directory in which you want WLST to write the properties files. This argument defaults to the pathname specified for the `scriptPath` argument.
createDeploymentScript	Optional. Boolean value specifying whether WLST creates a script that performs deployments only. This argument defaults to `false`, indicating that a deployment script is not created.

Example

The following example converts the configuration to a WLST script `config.py`. By default, the configuration file is loaded from `./config`, the script file is saved to `.config/config.py`, and the properties files is saved to `.config/config.py.properties`.

```
wls:/offline> configToScript()
configToScript is loading configuration from
c:\bea\user_projects\domains\wls\config\config.xml ...
Completed configuration load, now converting resources to wlst script...
configToScript completed successfully
The WLST script is written to c:\bea\user_projects\domains\wls\config\config.py
and the properties file associated with this script is written to
c:\bea\user_projects\domains\wls\config\config.py.properties
wls:/offline>
```

The following example converts server resources configured in the file `c:\bea\user_projects\domains\mydomain\config` directory to a WLST script `c:\bea\myscripts\config.py`.

```
wls:/offline> configToScript('c:/bea/user_projects/domains/mydomain',
'c:/bea/myscripts')
configToScript is loading configuration from
c:\bea\user_projects\domains\mydomain\config\config.xml ...
```

```
Completed configuration load, now converting resources to wlst script...
configToScript completed successfully
The WLST script is written to c:\bea\myscripts\config.py
and the properties file associated with this script is written to
c:\bea\mydomain\config.py.properties
wls:/offline>
```

dumpStack

Command Category: Information Commands
Use with WLST: Online or Offline

Description

Displays the stack trace from the last exception that occurred while performing a WLST action, and resets the stack trace.

If successful, the `dumpstack` command returns the Throwable object. In the event of an error, the command returns a `WLSTException`.

Syntax

```
dumpStack()
```

Example

This example displays the stack trace.

```
wls:/myserver/serverConfig> dumpStack()
com.bea.plateng.domain.script.jython.WLSTException:
java.lang.reflect.Invocation TargetException
...
```

dumpVariables

Command Category: Information Commands
Use with WLST: Online or Offline

Description

Displays all the variables used by WLST, including their name and value. In the event of an error, the command returns a `WLSTException`.

Syntax

```
dumpVariables()
```

Example

This example displays all the current variables and their values.

```
wls:/mydomain/serverConfig> dumpVariables()
adminHome   weblogic.rmi.internal.BasicRemoteRef - hostID:
   '-1 1080801509042639375:localhost:[7001,8001,-1,-1,-1,-1,-1]:
   mydomain:AdminServer', oid: '259', channel: 'null'
cmgr   [MBeanServerInvocationHandler]com.bea:Name=ConfigurationManager,
   Type=weblogic.management.mbeanservers.edit.ConfigurationManagerMBean
cmo   [MBeanServerInvocationHandler]com.bea:Name=mydomain,Type=Domain
connected true
domainName mydomain
...
wls:/mydomain/serverConfig>
```

find

Command Category: Information Commands
Use with WLST: Online

Description

Finds MBeans and attributes in the current hierarchy.

WLST returns the pathname to the MBean that stores the attribute and/or attribute type, and its value. If `searchInstancesOnly` is set to `false`, this command also searches the MBeanType paths that are not instantiated in the server, but that can be created. In the event of an error, the command returns a `WLSTException`.

Syntax

```
find([name], [type], [searchInstancesOnly])
```

Argument	Definition
name	Optional. Name of the attribute to find.

Argument	Definition (Continued)
type	Optional. Type of the attribute to find.
searchInstancesOnly	Optional. Boolean value specifying whether to search registered instances only or to also search MBeanTypes paths that are not instantiated in the server, but that can be created. This argument defaults to true, indicating only the registered instances will be searched.

Example

The following example searches for an attribute named javaCompiler in the current configuration hierarchy.

```
wls:/mydomain/serverConfig> find(name = 'JavaCompiler')
Finding 'JavaCompiler' in all registered MBean instances ...
/Servers/AdminServer                    JavaCompilerPreClassPath     null
/Servers/AdminServer                    JavaCompiler                 java
/Servers/AdminServer                     JavaCompilerPostClassPath   null
wls:/mydomain/serverConfig>
```

The following example searches for an attribute of type JMSRuntime in the current configuration hierarchy.

```
wls:/mydomain/serverRuntime> find(type='JMSRuntime')
Finding MBean of type 'JMSRuntime' in all the instances ...
/JMSRuntime/AdminServer.jms
wls:/mydomain/serverRuntime>
```

The following example searches for an attribute named execute in the current configuration hierarchy. The searchInstancesOnly argument is set to false, indicating to also search MBeanTypes that are not instantiated in the server.

```
wls:/mydomain/serverConfig> find(name='execute',
searchInstancesOnly='false')
Finding 'execute' in all registered MBean instances ...
/Servers/AdminServer        ExecuteQueues
[Ljavax.management.ObjectName;@1aa7dbc
/Servers/AdminSever         Use81StyleExecuteQueues
false
```

```
Now finding 'execute' in all MBean Types that can be instantiated ...
/Servers                                        ExecuteQueues
/Servers                                        Use81StyleExecuteQueues
wls:/mydomain/serverConfig>
```

getConfigManager

Command Category: Editing Commands
Use with WLST: Online

Description

Returns the latest `ConfigurationManager` MBean which manages the change process. You can then invoke methods to manage configuration changes across a domain. In the event of an error, the command returns a `WLSTException`.

Syntax

```
getConfigManager()
```

Example

The following example returns the latest `ConfigurationManagerBean` MBean and stores it within the task variable.

```
wls:/mydomain/serverConfig> cm=getConfigManager()
wls:/mydomain/serverConfig> cm=getType()
'weblogic.management.mbeanservers.edit.ConfigurationManagerMBean'
```

getMBean

Command Category: Information Commands
Use with WLST: Online

Description

Returns the MBean by browsing to the specified path. In the event of an error, the command returns a `WLSTException`.

Note: No exception is thrown if the MBean is not found.

Syntax

getMBean(*mbeanPath*)

Argument	Definition
mbeanPath	Path name to the MBean in the current hierarchy.

Example

The following example returns the MBean specified by the path.

```
wls:/mydomain/edit !> com=getMBean('Servers/myserver/COM/myserver')
wls:/mydomain/edit !> com.getType()
'Server'
```

getMBI

Command Category: Information Commands
Use with WLST: Online

Description

Returns the `MBeanInfo` for the specified `MBeanType` or the `cmo` variable. In the event of an error, the command returns a `WLSTException`.

Syntax

getMBI([*mbeanType*])

Argument	Definition
mbeanType	Optional. `MBeanType` for which the `MBeanInfo` is displayed.

Example

The following example gets the `MBeanInfo` for the specified `MBeanType` and stores it in the variable `svrMbi`.

```
wls:/mydomain/serverConfig>
svrMbi=getMBI('weblogic.management.configuration.ServerMBean')
```

getPath

Command Category: Information Commands
Use with WLST: Online

Description

Returns the MBean path for the specified MBean instance or ObjectName for the MBean in the current tree. In the event of an error, the command returns a `WLSTException`.

Syntax

`getPath(`*`mbean`*`)`

Argument	Definition
mbean	MBean instance or ObjectName for the MBean in the current tree for which you want to return the MBean path.

Example

The following example returns the MBean specified by the path.

```
wls:/mydomain/edit !> path=getPath('com.bea:Name=myserver,Type=Server')
wls:/mydomain/edit !> print path
'Servers/myserver'
```

listChildTypes

Command Category: Information Commands
Use with WLST: Online

Description

Lists all the child MBeans that can be created or deleted for the `cmo`. The `cmo` variable specifies the configuration bean instance to which you last navigated using WLST. For more information about the `cmo` variable, see "Changing the Current Management Object" on page 4-3.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
listChildTypes([parent])
```

Argument	Definition
parent	Optional. Parent type for which you want the children types listed.

Example

The following example lists the children MBeans that can be created or deleted for the `cmo` type.

```
wls:/mydomain/serverConfig> listChildTypes()
AppDeployments
BridgeDestinations
CachingRealms
Clusters
...
wls:/mydomain/serverConfig>
```

lookup

Command Category: Information Commands
Use with WLST: Online

Description

Looks up the specified MBean. The MBean must be a child of the current MBean. In the event of an error, the command returns a `WLSTException`.

Syntax

```
lookup(name, [childMBeanType])
```

Argument	Definition
name	Name of the MBean that you want to lookup.
childMBeanType	Optional. The type of the MBean that you want to lookup.

Example

The following example looks up the specified server, `myserver`, and stores the returned stub in the `sbean` variable.

```
wls:/mydomain/serverConfig> sbean=lookup('myserver','Server')
wls:/mydomain/serverConfig> sbean.getType()
'Server'
wls:/mydomain/serverConfig>
```

ls

Command Category: Information Commands
Use with WLST: Online or Offline

Description

Lists all the child beans and/or attributes for the current configuration or runtime bean.

You can optionally control the output by specifying an argument. If no argument is specified, the command lists all child beans and attributes in the domain. The output is returned as a string.

Note: Because WLST offline enables you to access and update the configuration objects that appear in the configuration files only, if you wish to view and/or change attribute values for a configuration object that is not already persisted in the configuration files as an XML element, you must first create the configuration object.

In the event of an error, the command returns a `WLSTException`.

Table A-8 describes the property information that appears in the left-hand column of the `ls` command output.

Note: The property information is based on the `MBeanInfo`; it does not reflect user permissions.

Table A-8 ls Command Output Information

Output	Definition
`d`	Configuration or runtime bean with which you can use the `cd` command; analogous to a directory in a UNIX or Windows file system.
`r`	Readable property.

Table A-8 ls Command Output Information (Continued)

Output	Definition (Continued)
w	Writeable property.
x	Executable operation.

Syntax

```
ls( ['a' | 'c' | 'o' | mbeanPath], [returnMap] [returnType])
```

Argument	Definition
a	Optional. Displays all the attribute names and values for the current configuration or runtime bean. If the attribute is encrypted, WLST displays six asterisks (******).
c	Optional. Displays all the child beans that are contained in the current configuration or runtime bean. This argument is the default.
o	Optional. This argument is only applicable when you are online (that is, WLST is connected to a running server). Displays all operations that can be invoked on the current runtime MBean. Operations are not shown for configuration MBeans.
mbeanPath	Optional. Path name to the MBean in the current hierarchy for which you want to list all child beans and attributes.
returnMap	Optional. Boolean value specifying whether to return a map containing the return values. If specified with the c argument, WLST returns a list of objects that contain all the children and referral MBean names. If specified with the a argument, WLST returns a HashMap of attribute name-value pairs. This argument defaults to false, indicating that no map is returned; in this case, WLST returns a String.
returnType	Optional. Controls the output returned in the map. This argument can be set to a, c, or o, and is only valid if *returnMap* is set to true. This argument defaults to c.

Example

The following example displays all the child configuration beans, and attribute names and values for the examples domain, which has been loaded into memory, in WLST offline mode:

```
wls:/offline/mydomain > ls()
dr--   AppDeployments
```

```
dr--   BridgeDestinations
dr--   Clusters
dr--   CustomResources
dr--   DeploymentConfiguration
dr--   Deployments
dr--   EmbeddedLDAP
dr--   ErrorHandlings
dr--   FileStores
dr--   InternalAppDeployments
dr--   InternalLibraries
dr--   JDBCDataSourceFactories
dr--   JDBCStores
dr--   JDBCSystemResources
dr--   JMSBridgeDestinations
dr--   JMSInteropModules
dr--   JMSServers
dr--   JMSSystemResources
dr--   JMX
...
wls:/offline/examples>
```

The following example displays all the attribute names and values in `DomainMBean`:

```
wls:/mydomain/serverConfig> ls('a')
-r--   AdminServerName                          AdminServer
-r--   AdministrationMBeanAuditingEnabled       false
-r--   AdministrationPort                       9002
-r--   AdministrationPortEnabled                false
-r--   AdministrationProtocol                   t3s
-r--   ArchiveConfigurationCount                0
-r--   ClusterConstraintsEnabled                false
-r--   ConfigBackupEnabled                      false
-r--   ConfigurationAuditType                   none
-r--   ConfigurationVersion                     9.0.0.0
-r--   ConsoleContextPath                       console
-r--   ConsoleEnabled                           true
-r--   ConsoleExtensionDirectory                console-ext
-r--   DomainVersion                            9.0.0.0
-r--   LastModificationTime                     0
```

```
-r--   Name                                    basicWLSDomain
-r--   Notes                                   null
-r--   Parent                                  null
-r--   ProductionModeEnabled                   false
-r--   RootDirectory                           .
-r--   Type                                    Domain
wls:/mydomain/serverConfig>
```

The following example displays all the child beans and attribute names and values in `Servers` MBean:

```
wls:/mydomain/serverConfig> ls('Servers')
dr--   AdminServer
```

The following example displays the attribute names and values for the specified MBean path and returns the information in a map:

```
wls:/mydomain/serverConfig> svrAttrList = ls('edit:/Servers/myserver',
returnMap='true', returnType='a')
-rw-   AcceptBacklog                           50
-rw-   AdminReconnectIntervalSeconds           10
-rw-   AdministrationPort                      9002
-rw-   AdministrationProtocol                  t3s
-rw-   AutoKillIfFailed                        false
-rw-   AutoMigrationEnabled                    false
-rw-   AutoRestart                             true
-rw-   COMEnabled                              false
-rw-   ClasspathServletDisabled                false
-rw-   ClientCertProxyEnabled                  false
-rw-   Cluster                                 null
-rw-   ClusterRuntime                          null
-rw-   ClusterWeight                           100
wls:/mydomain/serverConfig>
```

man

Command Category: Information Commands
Use with WLST: Online

Description

Displays help from `MBeanInfo` for the current MBean or its specified attribute. In the event of an error, the command returns a `WLSTException`.

Syntax

`man([attrName])`

Argument	Definition
attrName	Optional. MBean attribute name for which you would like to display help. If not specified, WLST displays helps for the current MBean.

Example

The following example displays help from `MBeanInfo` for the `ServerMBean` bean.

```
wls:/mydomain/serverConfig> man('Servers')
dynamic : true
creator : createServer
destroyer : destroyServer
description : <p>Returns the ServerMBeans representing the servers that have
been configured to be part of this domain.</p>
descriptorType : Attribute
Name : Servers
interfaceClassName : [Lweblogic.management.configuration.ServerMBean;
displayName : Servers
relationship : containment
```

redirect

Command Category: Information Commands
Use with WLST: Online or Offline

Description

Redirects WLST output to the specified filename.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
redirect(outputFile, [toStdOut])
```

Argument	Definition
outputFile	Name of the file to which you want to record the WLST commands. The filename can be absolute or relative to the directory from which you enter the command.
toStdOut	Optional. Boolean value specifying whether the output should be sent to `stdout`. This argument defaults to `true`, indicating that the output will be sent to `stdout`.

Example

The following example begins redirects WLST output to the `logs/wlst.log` file in the current directory:

```
wls:/mydomain/serverConfig> redirect('./logs/wlst.log')
```

removeListener

Command Category: Information Commands
Use with WLST: Online

Description

Removes a listener that was previously defined. If you do not specify an argument, WLST removes all listeners defined for all MBeans. For information about setting a listener, see "addListener" on page A-71.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
removeListener([mbean], [listenerName])
```

Argument	Definition
mbean	Optional. Name of the MBean or MBean object for which you want to remove the previously defined listeners.
listenerName	Optional. Name of the listener to be removed.

Example

The following example removes the listener named `mylistener`.

```
wls:/mydomain/serverConfig> removeListener(listenerName="mylistener")
```

showListeners

Command Category: Information Commands
Use with WLST: Online

Description

Shows all listeners that are currently defined. For information about setting a listener, see "addListener" on page A-71.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
showListeners()
```

Example

The following example shows all listeners that are currently defined.

```
wls:/mydomain/serverConfig> showListeners()
```

startRecording

Command Category: Information Commands
Use with WLST: Online or Offline

Description

Records all user interactions with WLST. This command is useful for capturing commands for replay.

In the event of an error, the command returns a `WLSTException`.

This command cannot be used when you are importing WLST as a Jython module, as described in "Importing WLST as a Jython Module" on page 2-11.

Syntax

```
startRecording(recordFile, [recordAll])
```

Argument	Definition
recordFile	Name of the file to which you want to record the WLST commands. The filename can be absolute or relative to the directory from which you invoked WLST.
recordAll	Optional. Boolean value specifying whether to capture all user interactions in the file. This argument defaults to `false`, indicating that only WLST commands are captured, and not WLST command output.

Example

The following example begins recording WLST commands in the `record.py` file:

```
wls:/mydomain/serverConfig> startRecording('c:/myScripts/record.py')
Starting recording to c:/myScripts/record.py
wls:/mydomain/serverConfig>
```

state

Command Category: Information Commands
Use with WLST: Online

Description

Using Node Manager, returns a map of servers or clusters and their state. Node Manager must be running.

For more information about server states, see "Understanding Server Life Cycle" in *Managing Server Startup and Shutdown*.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
state(name, [type])
```

Argument	Definition
name	Name of the server or cluster for which you want to retrieve the current state.
type	Optional. Type, `Server` or `Cluster`. This argument defaults to `Server`. When returning the state of a cluster, you must set this argument explicitly to `Cluster`, or the command will fail.

Example

The following example returns the state of the Managed Server, `managed1`.

```
wls:/mydomain/serverConfig> state('managed1','Server')
Current state of "managed1": SUSPENDED
wls:/mydomain/serverConfig>
```

The following example returns the state of the cluster, `mycluster`.

```
wls:/mydomain/serverConfig> state('mycluster','Cluster')
There are 3 server(s) in cluster: mycluster

States of the servers are
MServer1---SHUTDOWN
MServer2---SHUTDOWN
MServer3---SHUTDOWN
wls:/mydomain/serverConfig>
```

stopRecording

Command Category: Information Commands
Use with WLST: Online or Offline

Description

Stops recording WLST commands. For information about starting a recording, see "startRecording" on page A-87.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
stopRecording()
```

Example

The following example stops recording WLST commands.

```
wls:/mydomain/serverConfig> stopRecording()
Stopping recording to c:\myScripts\record.py
wls:/mydomain/serverConfig>
```

stopRedirect

Command Category: Information Commands
Use with WLST: Online or Offline

Description

Stops the redirection of WLST output to a file, if redirection is in progress.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
stopRedirect()
```

Example

The following example stops the redirection of WLST output to a file:

```
wls:/mydomain/serverConfig> stopRedirect()
WLST output will not be redirected to myfile.txt any more
```

storeUserConfig

Command Category: Information Commands
Use with WLST: Online

Description

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

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

In the event of an error, the command returns a `WLSTException`.

Syntax

```
storeUserConfig(userConfigFile, userKeyFile, [nm])
```

Argument	Definition
userConfigFile	Name of the file to store the user configuration. The filename can be absolute or relative to the directory from which you enter the command.
userKeyFile	Name of the file to store the key information that is associated with the user configuration file that you specify. The pathname can be absolute or relative to the directory from which you enter the command.
nm	Optional. Boolean value specifying whether to store the username and password for Node Manager or WebLogic Server. If set to true, the Node Manager username and password is stored. This argument default to `false`.

Example

The following example creates and stores a user configuration file and key file in the specified locations.

```
wls:/mydomain/serverConfig>
storeUserConfig('c:/myFiles/myuserconfigfile.secure',
'c:/myFiles/myuserkeyfile.secure')
Creating the key file can reduce the security of your system if it is not
kept in a secured location after it is created. Do you want to create the
key file? y or n
y
The username and password that were used for this current WLS connection are
stored in c:/myFiles/mysuserconfigfile.secure and
c:/myFiles/myuserkeyfile.secure
wls:/mydomain/serverConfig>
```

threadDump

Command Category: Information Commands
Use with WLST: Online or Offline

Description

Displays a thread dump for the specified server. In the event of an error, the command returns a `WLSTException`.

Syntax

```
threadDump([writeToFile], [fileName], [serverName])
```

Argument	Definition
writeToFile	Optional. Boolean value specifying whether to save the output to a file. This argument defaults to `true`, indicating that output is saved to a file.
fileName	Optional. Name of the file to which the output is written. The filename can be absolute or relative to the directory where WLST is running. This argument defaults to `Thread_Dump_`*serverName* file, where *serverName* indicates the name of the server. This argument is valid only if *writeToFile* is set to `true`.
serverName	Optional. Server name for which the thread dump is requested. This argument defaults to the server to which WLST is connected. If you are connected to an Administration Server, you can display a thread dump for the Administration Server and any Managed Server that is running in the domain. If you are connected to a Managed Server, you can only display a thread dump for that Managed Server.

Example

The following example displays the thread dump for the current server and saves the output to the `Thread_Dump_`*serverName* file.

```
wls:/mydomain/serverConfig> threadDump()
```

The following example displays the thread dump for the server `managedServer`. The information is not saved to a file.

```
wls:/mydomain/serverConfig> threadDump(writeToFile='false',
serverName='managedServer')
```

viewMBean

Command Category: Information Commands
Use with WLST: Online

Description

Displays information about an MBean, such as the attribute names and values, and operations. In the event of an error, the command returns a `WLSTException`.

Syntax

`viewMBean(`*`mbean`*`)`

Argument	Definition
mbean	MBean for which you want to display information.

Example

The following example displays information about the current MBean, `cmo`.

```
wls:/mydomain/serverConfig> cmo.getType()
'Domain'
wls:/mydomain/serverConfig> viewMBean(cmo)
Attribute Names and Values
--------------------------
XMLEntityCaches   null
Targets   javax.management.ObjectName[com.bea
:Name=MedRecJMSServer,Type=JMSServer,
   com.bea:Name=WSStoreForwardInternalJMSServerMedRecServer,Type=JMSServer,
   com.bea:Name=MedRecWseeJMSServer,Type=JMSServer,
   com.bea:Name=PhysWSEEJMSServer,Type=JMSServer,
   com.bea:Name=MedRecSAFAgent,Type=SAFAgent,
   com.bea:Name=AdminServer,Type=Server]
RootDirectory                                  .
EmbeddedLDAP
com.bea:Name=OOTB_medrec,Type=EmbeddedLDAP
RemoteSAFContexts  null
Libraries   javax.management.ObjectName[com.bea
...
wls:/mydomain/serverConfig>
```

writeIniFile

Command Category: Editing Commands
Use with WLST: Online

Description

Converts WLST definitions and method declarations to a Python (`.py`) file to enable advanced users to import them as a Jython module. After importing, the definitions and method declarations are available to other Jython modules and can be accessed directly using Jython syntax. For more information, see "Importing WLST as a Jython Module" on page 2-11.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
writeIniFile(filePath)
```

Argument	Definition
filePath	Full pathname to the file that you want to save the converted information.

Example

The following example converts WLST to a Python file named `wl.py`.

```
wls:/offline> writeIniFile("wl.py")
The Ini file is successfully written to wl.py
wls:/offline>
```

Life Cycle Commands

Use the WLST life cycle commands, listed in Table A-9, to manage the life cycle of a server instance.

For more information about the life cycle of a server instance, see "Understanding Server Life Cycle" in *Managing Server Startup and Shutdown*.

Table A-9 Life Cycle Commands for WLST Configuration

This command...	Enables you to...	Use with WLST...
"migrate" on page A-95	Migrate services to a target server within a cluster.	Online
"resume" on page A-97	Resume a server instance that is suspended or in `ADMIN` state.	Online
"shutdown" on page A-97	Gracefully shut down a running server instance or cluster.	Online
"start" on page A-100	Start a Managed Server instance or a cluster using Node Manager.	Online
"startServer" on page A-101	Start the Administration Server.	Online or Offline
"suspend" on page A-103	Suspend a running server.	Online

migrate

Command Category: Life Cycle Commands
Use with WLST: Online

Description

Migrates the specified services (JTA, JMS, or Server) to a targeted server within a cluster. In the event of an error, the command returns a `WLSTException`.

Syntax

```
migrate(sname, destinationName, [sourceDown], [destinationDown],
[migrationType])
```

Argument	Definition
sname	Name of the server from which the services should be migrated.
destinationName	Name of the machine or server to which you want to migrate the services.

Argument	Definition (Continued)
sourceDown	Optional. Boolean value specifying whether the source server is down. This argument defaults to `true`, indicating that the source server is not running. When migrating JTA services, the *sourceDown* argument is ignored, if specified, and defaults to `true`. The source server **must** be down in order for the migration of JTA services to succeed.
destinationDown	Optional. Boolean value specifying whether the destination server is down. This argument defaults to `false`, indicating that the destination server is running. If the destination is not running, and you do not set this argument to `true`, WLST returns a `MigrationException`. When migrating a JMS service to a non-running server instance, the server instance will activate the JMS service upon the next startup. When migrating the JTA Transaction Recovery Service to a non-running server instance, the target server instance will assume recovery services when it is started.
migrationType	Optional. Type of service(s) that you want to migrate. Valid values include: • `jta`—Migrate JTA services only. • `server`—Migrate Server services only. • `all`—Migrate all JTA and JMS services. This argument defaults to `all`.

Example

The following example migrates all JMS and JTA services on `server1` to the server `server2`. The boolean arguments specify that the source server is down and the destination server is running.

```
wls:/mydomain/edit !> migrate('server1','server2', 'true', 'false', 'all')
Migrating all JMS and JTA services from 'server1' to destination 'server2'
...
wls:/mydomain/edit !>
```

The following example migrates all Server services on `server1` to the server `server2`. The boolean arguments specify that the source server is down and the destination server is running.

```
wls:/mydomain/edit !> migrate('server1','server2', 'true', 'false',
'Server')
Migrating singleton server services from 'server1' to machine 'server2'...
wls:/mydomain/edit !>
```

resume

Command Category: Life Cycle Commands
Use with WLST: Online

Description

Resumes a server instance that is suspended or in ADMIN state. This command moves a server to the RUNNING state. For more information about server states, see "Understanding Server Life Cycle" in *Managing Server Startup and Shutdown*.

In the event of an error, the command returns a WLSTException.

Syntax

```
resume([sname], [block])
```

Argument	Definition
sname	Name of the server to resume. This argument defaults to the server to which WLST is currently connected.
block	Optional. Boolean value specifying whether WLST should block user interaction until the server is resumed. This argument defaults to false, indicating that user interaction is not blocked. In this case, WLST returns control to the user after issuing the command and assigns the task MBean associated with the current task to a variable that you can use to check its status. If you are importing WLST as a Jython module, as described in "Importing WLST as a Jython Module" on page 2-11, *block* is always set to true.

Example

The following example resumes a Managed Server instance.

```
wls:/mydomain/serverConfig> resume('managed1', block='true')
Server 'managed1' resumed successfully.
wls:/mydomain/serverConfig>
```

shutdown

Command Category: Life Cycle Commands
Use with WLST: Online

Description

Gracefully shuts down a running server instance or a cluster. The `shutdown` command waits for all the in-process work to be completed before shutting down the server or cluster.

You shut down a server to which WLST is connected by entering the `shutdown` command without any arguments.

When connected to a Managed Server instance, you only use the `shutdown` command to shut down the Managed Server instance to which WLST is connected; you cannot shut down another server while connected to a Managed Server instance.

WLST uses Node Manager to shut down a Managed Server. When shutting down a Managed Server, Node Manager must be running.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
shutdown([name], [entityType], [ignoreSessions], [timeOut], [force],
[block])
```

Argument	Definition
name	Optional. Name of the server or cluster to shutdown. This argument defaults to the server to which WLST is currently connected.
entityType	Optional. Type, `Server` or `Cluster`. This argument defaults to `Server`. When shutting down a cluster, you must set this argument explicitly to `Cluster`, or the command will fail.
ignoreSessions	Optional. Boolean value specifying whether WLST should drop all HTTP sessions immediately or wait for HTTP sessions to complete or timeout while shutting down. This argument defaults to `false`, indicating that all HTTP sessions must complete or timeout.
timeOut	Optional. Time (in milliseconds) that WLST waits for subsystems to complete in-process work and suspend themselves before shutting down the server. This argument defaults to 0 seconds, indicating that there is no timeout.
force	Optional. Boolean value specifying whether WLST should terminate a server instance or a cluster without waiting for the active sessions to complete. This argument defaults to `false`, indicating that all active sessions must complete before shutdown.

Argument	Definition (Continued)
block	Optional. Boolean value specifying whether WLST should block user interaction until the server is shutdown. This argument defaults to `false`, indicating that user interaction is not blocked. In this case, WLST returns control to the user after issuing the command and assigns the task MBean associated with the current task to a variable that you can use to check its status. If you are importing WLST as a Jython module, as described in "Importing WLST as a Jython Module" on page 2-11, *block* is always set to `true`.

Example

The following example instructs WLST to shutdown the server to which you are connected:

```
wls:/mydomain/serverConfig> shutdown()
Shutting down the admin server that you are currently connected to .......
Disconnected from weblogic server: AdminServer
```

The following example instructs WLST to wait 1000 seconds for HTTP sessions to complete or timeout (at 1000 ms) before shutting down `myserver`:

```
wls:/mydomain/serverConfig> shutdown('myserver','Server','false',1000,
block='false')
```

The following example instructs WLST to drop all HTTP sessions immediately while connected to a Managed Server instance:

```
wls:/mydomain/serverConfig> shutdown('MServer1','Server','true',1200)
Shutting down a managed server that you are connected to ...
Disconnected from weblogic server: MServer1
```

The following example instructs WLST to shutdown the cluster `mycluster`:

```
wls:/mydomain/serverConfig> shutdown('mycluster','Cluster')
Shutting down the cluster with name mycluster
Shutdown of cluster mycluster has been issued, please
refer to the logs to check if the cluster shutdown is successful.
Use the state(<server-name>) or state(<cluster-name>,"Cluster")
to check the status of the server or cluster
wls:/mydomain/serverConfig> state('mycluster','Cluster')
There are 3 server(s) in cluster: mycluster

States of the servers are
```

```
MServer1---SHUTDOWN
MServer2---SHUTDOWN
MServer3---SHUTDOWN
wls:/mydomain/serverConfig>
```

start

Command Category: Life Cycle Commands
Use with WLST: Online

Description

Starts a Managed Server instance or a cluster using Node Manager. WLST must be connected to the Administration Server and Node Manager must be running.

For more information about WLST commands used to connect to and use Node Manager, see "Node Manager Commands" on page A-104.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
start(name, [type], [url], [block])
```

Argument	Definition
name	Name of the Managed Server or cluster to start.
type	Optional. Type, `Server` or `Cluster`. This argument defaults to `Server`. When starting a cluster, you must set this argument explicitly to `Cluster`, or the command will fail.
url	Optional. Listen address and listen port of the server instance, specified using the following format: `[`*protocol*`://]`*listen-address*`:`*listen-port*. If not specified, this argument defaults to `t3://localhost:7001`.
block	Optional. Boolean value specifying whether WLST should block user interaction until the server or cluster is started. This argument defaults to `false`, indicating that user interaction is not blocked. In this case, WLST returns control to the user after issuing the command and assigns the task MBean associated with the current task to a variable that you can use to check its status. If you are importing WLST as a Jython module, as described in "Importing WLST as a Jython Module" on page 2-11, *block* is always set to `true`.

Example

The following example instructs Node Manager to start a Managed Server instance; the listen address is `localhost` and listen port is `8801`. WLST returns control to the user after issuing this command, as `block` is set to `false`.

```
wls:/mydomain/serverConfig> start('myserver', 'Server', block='false')
Starting server myserver ...
Server with name myserver started successfully.
wls:/mydomain/serverConfig>
```

The following example instructs Node Manager to start a cluster. WLST block user interaction until the cluster is started, as `block` defaults to `true`.

```
wls:/mydomain/serverConfig> start('mycluster', 'Cluster')
Starting the following servers in Cluster, mycluster: MS1, MS2, MS3...
.....................................................................
All servers in the cluster mycluster are started successfully.
wls:/mydomain/serverConfig>
```

startServer

Command Category: Life Cycle Commands
Use with WLST: Online or Offline

Description

Starts the Administration Server. In the event of an error, the command returns a `WLSTException`.

Syntax

```
startServer([adminServerName], [domainName], [url], [username], [password],
[domainDir], [block], [timeout], [serverLog], [systemProperties],
[jvmArgs])
```

Argument	Definition
adminServerName	Optional. Name of the Administration Server to start. This argument defaults to `myserver`.

Argument	Definition (Continued)
domainName	Optional. Name of the domain to which the Administration Server belongs. This argument defaults to `mydomain`.
url	Optional. URL of the Administration Server. This argument defaults to `t3://localhost:7001`.
username	Optional. Username use to connect WLST to the server. This argument defaults to `weblogic`.
password	Optional. Password used to connect WLST to the server. This argument defaults to `weblogic`.
domainDir	Optional. Domain directory in which the Administration Server is being started. This argument defaults to the current directory in which WLST is running.
block	Optional. Boolean value specifying whether WLST blocks user interaction until the server is started. When *block* is set to `false`, WLST returns control to the user after issuing the command and assigns the task MBean associated with the current task to a variable that you can use to check its status. This argument defaults to `true`, indicating that user interaction is blocked. If you are importing WLST as a Jython module, as described in "Importing WLST as a Jython Module" on page 2-11, *block* is always set to `true`.
timeout	Optional. Time (in milliseconds) that WLST waits for the server to start before canceling the operation. The default value is 60000 milliseconds. This argument is only applicable when *block* is set to `true`.
serverLog	Optional. Location of the server log file. This argument defaults to `stdout`.
systemProperties	Optional. System properties to pass to the server process. System properties should be specified as comma-seperated name-value pairs, and the name-value pairs should be separated by equals sign (=).
jvmArgs	Optional. JVM arguments to pass to the server process. Multiple arguments can be specified, separated by commas.

Example

The following example starts the Administration Server named `demoServer` in the `demoDomain`.

```
wls:/offline> startServer('demoServer','demoDomain','t3://localhost:8001',
'myweblogic','wlstdomain','c:/mydomains/wlst','false', 60000,
```

```
jvmArgs='-XX:MaxPermSize=75m, -Xmx512m, -XX:+UseParallelGC')
wls:/offline>
```

suspend

Command Category: Life Cycle Commands
Use with WLST: Online

Description

Suspends a running server. This command moves a server from the `RUNNING` state to the `ADMIN` state. For more information about server states, see "Understanding Server Life Cycle" in *Managing Server Startup and Shutdown*.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
suspend([sname], [ignoreSessions], [timeOut], [force], [block])
```

Argument	Definition
sname	Optional. Name of the server to suspend. The argument defaults to the server to which WLST is currently connected.
ignoreSessions	Optional. Boolean value specifying whether WLST should drop all HTTP sessions immediately or wait for HTTP sessions to complete or time out while suspending. This argument defaults to `false`, indicating that HTTP sessions must complete or time out.
timeOut	Optional. Time (in seconds) the WLST waits for the server to complete in-process work before suspending the server. This argument defaults to 0 seconds, indicating that there is no timeout.
force	Optional. Boolean value specifying whether WLST should suspend the server without waiting for active sessions to complete. This argument defaults to `false`, indicating that all active sessions must complete before suspending the server.
block	Optional. Boolean value specifying whether WLST blocks user interaction until the server is started. This argument defaults to `false`, indicating that user interaction is not blocked. In this case, WLST returns control to the user after issuing the command and assigns the task MBean associated with the current task to a variable that you can use to check its status. If you are importing WLST as a Jython module, as described in "Importing WLST as a Jython Module" on page 2-11, *block* is always set to `true`.

Example

The following example suspends a Managed Server instance:

```
wls:/mydomain/serverConfig> suspend('managed1')
Server 'managed1' suspended successfully.
wls:/mydomain/serverConfig>
```

Node Manager Commands

Use the WLST Node Managers commands, listed in Table A-10, to start, shut down, restart, and monitor WebLogic Server instances. For more information about Node Manager, see "Using Node Manager to Control Servers" in *Managing Server Startup and Shutdown*.

Note: Node Manager must be running before you can execute the commands within this category. For all commands except for `nm` and `nmconnect`, WLST must be connected to a running Node Manager.

Table A-10 Node Manager Commands for WLST Configuration

This command...	Enables you to...	Use with WLST...
"nm" on page A-105	Determine whether WLST is connected to Node Manager.	Online
"nmConnect" on page A-106	Connect WLST to Node Manager to establish a session.	Online or Offline
"nmDisconnect" on page A-108	Disconnect WLST from a Node Manager session.	Online
"nmEnroll" on page A-108	Enroll the machine on which WLST is currently running.	Online
"nmGenBootStartupProps" on page A-110	Generates the Node Manager property files, `boot.properties` and `startup.properties`, for the specified server.	Online
"nmKill" on page A-111	Kill the specified server instance that was started with Node Manager.	Online
"nmLog" on page A-111	Return the Node Manager log.	Online
"nmServerLog" on page A-112	Return the server output log of the server that was started with Node Manager.	Online
"nmServerStatus" on page A-113	Return the status of the server that was started with Node Manager.	Online

Table A-10 Node Manager Commands for WLST Configuration (Continued)

This command...	Enables you to...	Use with WLST...
"nmStart" on page A-114	Start a server in the current domain using Node Manager.	Online
"nmVersion" on page A-115	Return the Node Manager server version.	Online
"startNodeManager" on page A-115	Start Node Manager at default port (5556).	Online or Offline

nm

Command Category: Node Manager Commands
Use with WLST: Online

Description

Determines whether WLST is connected to Node Manager. Returns `true` or `false` and prints a descriptive message. Node Manager must be running before you can execute this command.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
nm()
```

Example

The following example indicates that WLST is currently connected to Node Manager that is monitoring `mydomain`.

```
wls:/mydomain/serverConfig> nm()
Currently connected to Node Manager that is monitoring the domain "mydomain"
wls:/mydomain/serverConfig>
```

The following example indicates that WLST is not currently connected to Node Manager.

```
wls:/mydomain/serverConfig> nm()
Not connected to any Node Manager
wls:/mydomain/serverConfig>
```

nmConnect

Command Category: Node Manager Commands
Use with WLST: Online or Offline

Description

Connects WLST to Node Manager to establish a session. After connecting to Node Manager, you can invoke any Node Manager commands via WLST. Node Manager must be running before you can execute this command.

Once connected, the WLST prompt displays as follows, where *domainName* indicates the name of the domain that is being managed: `wls:/nm/domainName>`. If you then connect WLST to a WebLogic Server instance, the prompt is changed to reflect the WebLogic Server instance. You can use the `nm` command to determine whether WLST is connected to Node Manager, as described in "nm" on page A-105.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
nmConnect([username, password], [host], [port], [domainName], [domainDir]
[nmType], [verbose])
```

```
nmConnect([userConfigFile, userKeyFile], [host], [port], [domainName],
[domainDir] [nmType], [verbose])
```

Argument	Definition
username	Username of the operator who is connecting WLST to Node Manager. The username defaults to `weblogic`. **Note:** When running a server in production mode, you must specify the username and password explicitly on the command line to ensure that the appropriate username and password are used when connecting to Node Manager.
password	Password of the operator who is connecting WLST to Node Manager. The password defaults to `weblogic`. **Note:** When running a server in production mode, you must specify the username and password explicitly on the command line to ensure that the appropriate username and password are used when connecting to Node Manager.
host	Optional. Host name of Node Manager. This argument defaults to `localhost`.

Argument	Definition (Continued)
port	Optional. Port number of Node Manager. This argument defaults to a value that is based on the Node Manager server type, as follows: • For `plain` type, defaults to 5556 • For `rsh` type, defaults to 514 • For `ssh` type, defaults to 22 • For `ssl` type, defaults to 5556
domainName	Optional. Name of the domain that you want to manage. This argument defaults to `mydomain`.
domainDir	Optional. Path of the domain directory to which you want to save the Node Manager secret file (`nm_password.properties`) and `SerializedSystemIni.dat` file. This argument defaults to the directory in which WLST was started.
nmType	Type of the Node Manager server. Valid values include: • `plain` for plain socket Java-based implementation • `rsh` for RSH implementation • `ssh` for script-based SSH implementation • `ssl` for Java-based SSL implementation This argument defaults to `ssl`.
verbose	Optional. Boolean value specifying whether WLST connects to Node Manager in verbose mode. This argument defaults to `false`, disabling verbose mode.
userConfigFile	Optional. Name and location of a user configuration file which contains an encrypted username and password. When you create a user configuration file, the `storeUserConfig` command uses a key file to encrypt the username and password. Only the key file that encrypts a user configuration file can decrypt the username and password. (See "storeUserConfig" on page A-90.)
userKeyFile	Optional. Name and location of the key file that is associated with the specified user configuration file and is used to decrypt it. (See "storeUserConfig" on page A-90.)

Example

The following example connects WLST to Node Manager to monitor the `oamdomain` domain using the default host and port numbers and `plain` Node Manager type.

```
wls:/myserver/serverConfig> nmConnect('weblogic', 'weblogic', 'localhost',
'5555', 'oamdomain', 'c:/bea/user_projects/domains/oamdomain','plain')
Connecting to Node Manager Server ...
Successfully connected to Node Manager.
wls:/nm/oamdomain>
```

nmDisconnect

Command Category: Node Manager Commands
Use with WLST: Online

Description

Disconnects WLST from a Node Manager session. WLST must be connected to Node Manager to run this command.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
nmDisconnect()
```

Example

The following example disconnects WLST from a Node Manager session.

```
wls:/nm/oamdomain> nmDisconnect()
Successfully disconnected from Node Manager
wls:/myserver/serverConfig>
```

nmEnroll

Command Category: Node Manager Commands
Use with WLST: Online

Description

Enrolls the machine on which WLST is currently running. WLST must be connected to an Administration Server to run this command; WLST does not need to be connected to Node Manager.

This command downloads the following files from the Administration Server:

- Node Manager secret file (`nm_password.properties`), which contains the encrypted username and password that is used for server authentication
- `SerializedSystemIni.dat` file

This command also updates the `nodemanager.domains` file under the *`WL_HOME`*`/common/nodemanager` directory with the domain information, where *`WL_HOME`* refers to the top-level installation directory for WebLogic Server.

You must run this command once per domain per machine unless that domain shares the root directory of the Administration Server.

If the machine is already enrolled when you run this command, the Node Manager secret file (`nm_password.properties`) is refreshed with the latest information from the Administration Server.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
nmEnroll([domainDir], [nmHome])
```

Argument	Definition
domainDir	Optional. Path of the domain directory to which you want to save the Node Manager secret file (`nm_password.properties`) and `SerializedSystemIni.dat` file. This argument defaults to the directory in which WLST was started.
nmHome	Optional. Path to the Node Manager home. The `nodemanager.domains` file, containing the domain information, is written to this directory. This argument defaults to *`WL_HOME`*`/common/nodemanager`, where *`WL_HOME`* refers to the top-level installation directory for WebLogic Server.

Example

The following example enrolls the current machine with Node Manager and saves the Node Manager secret file (`nm_password properties`) and `SerializedSystemIni.dat` file to `c:/bea/mydomain/common/nodemanager/nm_password.properties`. The `nodemanager.domains` file is written to *`WL_HOME`*`/common/nodemanager` by default.

```
wls:/mydomain/serverConfig> nmEnroll('c:/bea/mydomain/common/nodemanager')
Enrolling this machine with the domain directory at
c:\bea\mydomain\common\nodemanager....
```

```
Successfully enrolled this machine with the domain directory at
C:\bea\mydomain\common\nodemanager
wls:/mydomain/serverConfig>
```

nmGenBootStartupProps

Command Category: Node Manager Commands
Use with WLST: Online

Description

Generates the Node Manager property files, `boot.properties` and `startup.properties`, for the specified server. The Node Manager property files are stored relative to the root directory of the specified server. The target root directory must be on the same machine on which you are running the command.

You must specify the name of a server; otherwise, the command will fail.

In the event of an error, the command returns a `WLSTException`.

Syntax

`nmGenBootStartupProps(`*`serverName`*`)`

Argument	Definition
serverName	Name of the server for which Node Manager property files are generated.

Example

The following example generates `boot.properties` and `startup.properties` in the root directory of the specified server, `ms1`.

```
wls:/mydomain/serverConfig> nmGenBootStartupPoprs('ms1')
Successfully generated boot.properties at
c:\bea\mydomain\servers\ms1\data\nodemanager\boot.properties
Successfully generated startup.properties at
c:\bea\mydomain\servers\ms1\data\nodemanager\startup.properties
wls:/mydomain/serverConfig>
```

nmKill

Command Category: Node Manager Commands
Use with WLST: Online

Description

Kills the specified server instance that was started with Node Manager.

If you do not specify a server name using the *serverName* argument, the argument defaults to `myServer`, which must match your server name or the command will fail.

If you attempt to kill a server instance that was not started using Node Manager, the command displays an error. WLST must be connected to Node Manager to run this command.

In the event of an error, the command returns a `WLSTException`.

Syntax

`nmKill([serverName])`

Argument	Definition
serverName	Optional. Name of the server to be killed. This argument defaults to `myserver`.

Example

The following example kills the server named `oamserver`.

```
wls:/nm/oamdomain> nmKill('oamserver')
Killing server 'oamserver' ...
Server oamServer killed successfully.
wls:/nm/oamdomain>
```

nmLog

Command Category: Node Manager Commands
Use with WLST: Online

Description

Returns the Node Manager log. WLST must be connected to Node Manager to run this command.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
nmLog([writer])
```

Argument	Definition
writer	Optional. `java.io.Writer` object to which you want to stream the log output. This argument defaults to the WLST writer stream.

Example

The following example displays the Node Manager log.

```
wls:/nm/oamdomain> nmLog()
Successfully retrieved the Node Manager log and written.
wls:/nm/oamdomain>
```

nmServerLog

Command Category: Node Manager Commands
Use with WLST: Online

Description

Returns the server output log of the server that was started with Node Manager. WLST must be connected to Node Manager to run this command.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
nmServerLog([serverName], [writer])
```

Argument	Definition
serverName	Optional. Name of the server for which you want to display the server output log. This argument defaults to `myserver`.

Argument	Definition (Continued)
writer	Optional. `java.io.Writer` object to which you want to stream the log output. This argument defaults to the WLSTInterpreter standard out, if not specified.

Example

The following example displays the server output log for the `oamserver` server and writes the log output to `myWriter`.

```
wls:/nm/oamdomain> nmServerLog('oamserver',myWriter)
Successfully retrieved the server log and written.
wls:/nm/oamdomain>
```

nmServerStatus

Command Category: Node Manager Commands
Use with WLST: Online

Description

Returns the status of the server that was started with Node Manager. WLST must be connected to Node Manager to run this command.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
nmServerStatus([serverName])
```

Argument	Definition
serverName	Optional. Name of the server for which you want to display the status. This argument defaults to `myserver`.

Example

The following example displays the status of the server named `oamserver`, which was started with Node Manager.

```
wls:/nm/oamdomain> nmServerStatus('oamserver')
RUNNING
wls:/nm/oamdomain>
```

nmStart

Command Category: Node Manager Commands
Use with WLST: Online

Description

Starts a server in the current domain using Node Manager. WLST must be connected to Node Manager to run this command.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
nmStart([serverName], [domainDir], [props], [writer])
```

Argument	Definition
serverName	Optional. Name of the server to be started.
domainDir	Optional. Domain directory of the server to be started. This argument defaults to the current directory in which WLST is running.
props	Optional. System properties to apply to the new server.
writer	Optional. `java.io.Writer` object to which the server output is written. This argument defaults to the WLST writer.

Example

The following example starts the `managed1` server in the current domain using Node Manager.

```
wls:/nm/mydomain> nmStart("managed1")
Starting server managed1 ...
Server managed1 started successfully
wls:/nm/mydomain>
```

The following example starts the Administration Server in the specified domain using Node Manager. In this example, the `prps` variable stores the system property settings and is passed to the command using the `props` argument.

```
wls:/nm/mydomain> prps = makePropertiesObject("weblogic.ListenPort=8001")
wls:/nm/mydomain> nmStart("AdminServer",props=prps)
Starting server AdminServer...
Server AdminServer started successfully
wls:/nm/mydomain>
```

nmVersion

Command Category: Node Manager Commands
Use with WLST: Online

Description

Returns the Node Manager server version. WLST must be connected to Node Manager to run this command.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
nmVersion()
```

Example

The following example displays the Node Manager server version.

```
wls:/nm/oamdomain> nmVersion()
The Node Manager version that you are currently connected to is 9.0.0.0
wls:/nm/oamdomain>
```

startNodeManager

Command Category: Node Manager Commands
Use with WLST: Online or Offline

Description

Starts Node Manager at the default port (5556).

Note: The WebLogic Server custom installation process optionally installs and starts Node Manager as a Windows service on Windows systems. For more information, see "About Installing Node Manager as a Windows Service" in *BEA Products Installation Guide*. In this case, you do not need to start the Node Manager manually.

If Node Manager is already running when you invoke the `startNodeManager` command, the following message is displayed:

```
A Node Manager has already been started.
Cannot start another Node Manager process via WLST
```

In the event of an error, the command returns a `WLSTException`.

Syntax

```
startNodeManager([verbose], [nmProperties])
```

Example

Argument	Definition
verbose	Optional. Boolean value specifying whether WLST starts Node Manager in verbose mode. This argument defaults to `false`, disabling verbose mode.
nmProperties	Optional. Comma-separated list of Node Manager properties, specified as name-value pairs. Node Manager properties include, but are not limited to, the following: `NodeManagerHome`, `ListenAddress`, `ListenPort`, and `PropertiesFile`.

The following example displays the Node Manager server version.

```
wls:/mydomain/serverConfig> startNodeManager(verbose='true',
NodemanagerHome='c:/bea/weblogic91/common/nodemanager', ListenPort='6666',
ListenAddress='myhost'))
Launching Node Manager ...
Successfully launched the Node Manager.
The Node Manager process is running independent of the WLST process
Exiting WLST will not stop the Node Manager process. Please refer
to the Node Manager logs for more information.
The Node Manager logs will be under c:\bea\weblogic91\common\nodemanager.
wls:/mydomain/serverConfig>
```

Tree Commands

Use the WLST tree commands, listed in Table A-11, to navigate among MBean hierarchies.

Table A-11 Tree Commands for WLST Configuration

Use this command...	To...	Use with WLST...
"config" on page A-118	Navigate to the last MBean to which you navigated in the configuration MBean hierarchy or to the root of the hierarchy, `DomainMBean`. **Note:** This command is deprecated as of WebLogic Server 9.0. You should update your script to use the `serverConfig` command as described in "serverConfig" on page A-125.	Online
"custom" on page A-119	Navigate to the root of custom MBeans that are registered in the server.	Online
"domainConfig" on page A-120	Navigate to the last MBean to which you navigated in the domain configuration hierarchy or to the root of the hierarchy, `DomainMBean`.	Online
"domainRuntime" on page A-121	Navigate to the last MBean to which you navigated in the domain runtime hierarchy or to the root of the hierarchy, `DomainRuntimeMBean`.	Online
"edit" on page A-123	Navigate to the last MBean to which you navigated in the edit configuration MBean hierarchy or to the root of the hierarchy, `DomainMBean`.	Online
"jndi" on page A-124	Navigates to the JNDI tree for the server to which WLST is currently connected.	Online
"runtime" on page A-124	Navigate to the last MBean to which you navigated in the runtime hierarchy or the root of all runtime objects, `DomainRuntimeMBean`. **Note:** This command is deprecated as of WebLogic Server 9.0. You should update your scripts to use the `serverRuntime` command, as described in "serverRuntime" on page A-126.	Online

Table A-11 Tree Commands for WLST Configuration (Continued)

Use this command...	To...	Use with WLST...
"serverConfig" on page A-125	Navigate to the last MBean to which you navigated in the configuration MBean hierarchy or to the root of the hierarchy, `DomainMBean`.	Online
"serverRuntime" on page A-126	Navigate to the last MBean to which you navigated in the runtime MBean hierarchy or to the root of the hierarchy, `ServerRuntimeMBean`.	Online

config

Command Category: Tree Commands
Use with WLST: Online

Description

Note: This command is deprecated as of WebLogic Server 9.0. You should update your scripts to use the `serverConfig` command, as described in "serverConfig" on page A-125.

Navigates to the last MBean to which you navigated in the configuration MBean hierarchy or to the root of the hierarchy, `DomainMBean`. For more information, see "Navigating Among MBean Hierarchies" on page 4-9.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
config()
```

Example

The following example illustrates how to navigate from the runtime MBean hierarchy to the configuration MBean hierarchy on an Administration Server instance:

```
wls:/mydomain/runtime> config()
Location changed to config tree (deprecated). This is a writeable tree with
DomainMBean as the root.
For more help, use help('config')
wls:/mydomain/config> ls()
```

```
dr--   Applications
dr--   BridgeDestinations
dr--   Clusters
dr--   DeploymentConfiguration
dr--   Deployments
dr--   DomainLogFilters
dr--   EmbeddedLDAP
dr--   JDBCConnectionPools
dr--   JDBCDataSourceFactories
dr--   JDBCDataSources
dr--   JDBCMultiPools
dr--   JDBCTxDataSources
dr--   JMSBridgeDestinations
dr--   JMSConnectionFactories
dr--   JMSDestinationKeys
dr--   JMSDestinations
dr--   JMSDistributedQueueMembers
dr--   JMSDistributedQueues
dr--   JMSDistributedTopicMembers
dr--   JMSDistributedTopics
dr--   JMSFileStores
dr--   JMSJDBCStores
...
wls:/mydomain/config>
```

custom

Command Category: Tree Commands
Use with WLST: Online

Description

Navigates to the root of custom MBeans that are registered in the server. WLST navigates, interrogates, and edits custom MBeans as it does domain MBeans; however, custom MBeans cannot use the `cmo` variable because a stub is not available.

Note: When navigating to the `custom` tree, WLST queries all MBeans in the compatibility MBean server, the runtime MBean server, and potentially the JVM platform MBean server to locate the custom MBeans. Depending on the number of MBeans in the current

domain, this process make take a few minutes, and WLST may not return a prompt right away.

The `custom` command is available when WLST is connected to an Administration Server instance or a Managed Server instance. When connected to a WebLogic Integration or WebLogic Portal server, WLST can interact with all the WebLogic Integration or WebLogic Portal server MBeans.

For more information about custom MBeans, see *Developing Custom Management Utilities with JMX*.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
custom()
```

Example

The following example navigates from the configuration MBean hierarchy to the custom MBean hierarchy on a Administration Server instance.

```
wls:/mydomain/serverConfig> custom()
Location changed to custom tree. This is a writeable tree with No root. For
more help, use help('custom')
wls:/mydomain/custom>
```

domainConfig

Command Category: Tree Commands
Use with WLST: Online

Description

Navigates to the last MBean to which you navigated in the domain Configuration hierarchy or to the root of the hierarchy, `DomainMBean`. This read-only hierarchy stores the configuration MBeans that represent your current domain.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
domainConfig()
```

Example

The following example navigates from the configuration MBean hierarchy to the domain Configuration hierarchy on an Administration Server instance.

```
wls:/mydomain/serverConfig> domainConfig()
Location changed to domainConfig tree. This is a read-only tree with
DomainMBean as the root.
For more help, use help('domainConfig')
wls:/mydomain/domainConfig> ls()
dr--   AppDeployments
dr--   BridgeDestinations
dr--   Clusters
dr--   CustomResources
dr--   DeploymentConfiguration
dr--   Deployments
dr--   EmbeddedLDAP
dr--   ErrorHandlings
dr--   FileStores
dr--   InternalAppDeployments
dr--   InternalLibraries
dr--   JDBCDataSourceFactories
dr--   JDBCStores
dr--   JDBCSystemResources
dr--   JMSBridgeDestinations
dr--   JMSInteropModules
dr--   JMSServers
dr--   JMSSystemResources
...
wls:/mydomain/domainConfig>
```

domainRuntime

Command Category: Tree Commands
Use with WLST: Online

Description

Navigates to the last MBean to which you navigated in the domain Runtime hierarchy or to the root of the hierarchy, `DomainRuntimeMBean`. This read-only hierarchy stores the runtime MBeans that represent your current domain.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
domainRuntime()
```

Example

The following example navigates from the configuration MBean hierarchy to the domain Runtime hierarchy on an Administration Server instance.

```
wls:/mydomain/serverConfig> domainRuntime()
wls:/mydomain/domainRuntime> ls()
dr--   AppRuntimeStateRuntime
dr--   DeployerRuntime
dr--   DomainServices
dr--   LogRuntime
dr--   MessageDrivenControlEJBRuntime
dr--   MigratableServiceCoordinatorRuntime
dr--   MigrationDataRuntimes
dr--   SNMPAgentRuntime
dr--   ServerLifeCycleRuntimes
dr--   ServerRuntimes
dr--   ServerServices

-r--   ActivationTime                          Mon Aug 01 11:41:25 EDT 2005
-r--   Clusters                                     null
-r--   MigrationDataRuntimes                        null
-r--   Name                                         sampleMedRecDomain
-rw-   Parent                                       null
-r--   SNMPAgentRuntime                             null
-r--   Type                                         DomainRuntime
-r-x   restartSystemResource                        Void :
WebLogicMBean(weblogic.management.configuration.SystemResourceMBean)
wls:/mydomain/domainRuntime>
```

edit

Command Category: Tree Commands
Use with WLST: Online

Description

Navigates to the last MBean to which you navigated in the edit configuration MBean hierarchy or to the root of the hierarchy, `DomainMBean`. This writeable hierarchy stores all of the configuration MBeans that represent your current domain.

Note: To edit configuration beans, you must be connected to an Administration Server. If you connect to a Managed Server, WLST functionality is limited to browsing the configuration bean hierarchy. While you cannot use WLST to change the values of MBeans on Managed Servers, it is possible to use the Management APIs to do so. BEA Systems recommends that you change only the values of configuration MBeans on the Administration Server. Changing the values of MBeans on Managed Servers can lead to an inconsistent domain configuration.

For more information about editing configuration beans, see "Editing Configuration MBeans" on page 4-12.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
edit()
```

Example

The following example illustrates how to navigate from the server configuration MBean hierarchy to the editable copy of the domain configuration MBean hierarchy, in an Administration Server instance.

```
wls:/myserver/serverConfig> edit()
Location changed to edit tree. This is a writeable tree with DomainMBean as
the root.
For more help, use help('edit')
wls:/myserver/edit !> ls()
dr--   AppDeployments
dr--   BridgeDestinations
dr--   Clusters
dr--   DeploymentConfiguration
```

```
dr--   Deployments
dr--   EmbeddedLDAP
...
wls:/myserver/edit !>
```

jndi

Command Category: Tree Commands
Use with WLST: Online

Description

Navigates to the JNDI tree for the server to which WLST is currently connected. This read-only tree holds all the elements that are currently bound in JNDI.

In the event of an error, the command returns a WLSTException.

Syntax

```
jndi()
```

Example

The following example navigates from the runtime MBean hierarchy to the Domain JNDI tree on an Administration Server instance.

```
wls:/myserver/runtime> jndi()
Location changed to jndi tree. This is a read-only tree with No root. For
more help, use help('jndi')
wls:/myserver/jndi> ls()
dr--   ejb
dr--   javax
dr--   jms
dr--   weblogic
...
```

runtime

Command Category: Tree Commands
Use with WLST: Online

Description

Note: This command is deprecated as of WebLogic Server 9.0. You should update your scripts to use the `serverRuntime` command, as described in "serverRuntime" on page A-126.

Navigates to the last MBean to which you navigated in the runtime hierarchy or the root of all runtime objects, `DomainRuntimeMBean`. When connected to a Managed Server instance, the root of runtime MBeans is `ServerRuntimeMBean`.

In the event of an error, the command returns a `WLSTException`.

For more information, see "Browsing Runtime MBeans" on page 4-6.

Syntax

```
runtime()
```

Example

The following example navigates from the configuration MBean hierarchy to the runtime MBean hierarchy on a Managed Server instance.

```
wls:/mydomain/serverConfig> runtime()
Location changed to runtime tree (deprecated). This is a read-only tree with
DomainRuntimeMBean as the root.
For more help, use help('runtime')
wls:/mydomain/runtime>
```

serverConfig

Command Category: Tree Commands
Use with WLST: Online

Description

Navigates to the last MBean to which you navigated in the configuration MBean hierarchy or to the root of the hierarchy, `DomainMBean`.

This read-only hierarchy stores the configuration MBeans that represent the server to which WLST is currently connected. The MBean attribute values include any command-line overrides that a user specified while starting the server.

In the event of an error, the command returns a `WLSTException`.

For more information, see "Navigating Among MBean Hierarchies" on page 4-9.

Syntax

```
serverConfig()
```

Example

The following example navigates from the domain runtime MBean hierarchy to the configuration MBean hierarchy on an Administration Server instance.

```
wls:/mydomain/domainRuntime> serverConfig()
wls:/mydomain/serverConfig>
```

serverRuntime

Command Category: Tree Commands
Use with WLST: Online

Description

Navigates to the last MBean to which you navigated in the runtime MBean hierarchy or to the root of the hierarchy, `ServerRuntimeMBean`. This read-only hierarchy stores the runtime MBeans that represent the server to which WLST is currently connected.

In the event of an error, the command returns a `WLSTException`.

Syntax

```
serverRuntime()
```

Example

The following example navigates from the configuration MBean hierarchy to the runtime MBean hierarchy on an Administration Server instance.

```
wls:/mydomain/serverConfig> serverRuntime()
Location changed to serverRuntime tree. This is a read-only tree with
ServerRuntimeMBean as the root.
For more help, use help('serverRuntime')
wls:/mydomain/serverRuntime>
```

WLST Variable Reference

Table A-12 describes WLST variables and their common usage. All variables are initialized to default values at the start of a user session and are changed according to the user interaction with WLST.

Table A-12 WLST Variables

Variable	Description	Example
`adminHome`	`Administration` MBean. This variable is available only when WLST is connected to the Administration Server. **Note:** This variable is deprecated as of WebLogic Server 9.0.	`wls:/mydomain/edit> **bean = adminHome.getMBean(ObjectName('mydomain:Name=mydomain,Type=Domain'))**`
`cmo`	Current Management Object. The `cmo` variable is set to the configuration bean instance to which you navigate using WLST. You use this variable to perform any `create`, `get`, `set`, or `invoke` method on the current configuration bean instance. By default, this variable is initialized to the root of all configuration management objects, `DomainMBean`.	`wls:/mydomain/edit> **cmo.getAdministrationPort()**` `9002`
`connected`	Boolean value specifying whether WLST is connected to a running server. WLST sets this variable to `true` when connected to a running server; otherwise, WLST sets it to `false`.	`wls:/mydomain/serverConfig> **print connected**` `false`
`domainName`	Name of the domain to which WLST is connected.	`wls:/mydomain/serverConfig> **print domainName**` `mydomain`
`domainRuntimeService`	`DomainRuntimeServiceMBean` MBean. This variable is available only when WLST is connected to the Administration Server.	`wls:/mydomain/serverConfig> **domainService.getServerName()**` `'myserver'`
`editService`	`EditServiceMBean` MBean. This variable is available only when WLST is connected to the Administration Server.	`wls:/mydomain/edit> **dc = editService.getDomainConfiguration()**`

Table A-12 WLST Variables (Continued)

Variable	Description	Example
`exitonerror`	Boolean value specifying whether WLST terminates script execution when it encounters an exception. This variable defaults to `true`, indicating that script execution is terminated when WLST encounters an error. This variable is not applicable when running WLST in interactive mode.	`wls:/mydomain/serverConfig>` **`print exitonerror`** `true`
`home`	`Local` MBean. **Note:** This variable is deprecated as of WebLogic Server 9.0.	`wls:/mydomain/serverConfig>` **`bean = home.getMBean(ObjectName('mydomain:Name=mydomain,Type=Domain'))`**
`isAdminServer`	Boolean value specifying whether WLST is connected to a WebLogic Administration Server instance. WLST sets this variable to `true` if WLST is connected to a WebLogic Administration Server; otherwise, WLST sets it to `false`.	`wls:/mydomain/serverConfig>` **`print isAdminServer`** `true`
`mbs`	`MBeanServerConnection` object that corresponds to the current location in the hierarchy.	`wls:/mydomain/serverConfig>` **`mbs.isRegistered(ObjectName('mydomain:Name=mydomain,Type=Domain'))`**
`recording`	Boolean value specifying whether WLST is recording commands. WLST sets this variable to `true` when the `startRecording` command is entered; otherwise, WLST sets this variable to `false`.	`wls:/mydomain/serverConfig>` **`print recording`** `true`
`runtimeService`	`RuntimeServiceMBean` MBean.	`wls:/mydomain/serverConfig>` **`sr=runtimeService.getServerRuntime()`**
`serverName`	Name of the server to which WLST is connected.	`wls:/mydomain/serverConfig>` **`print serverName`** `myserver`

Table A-12 WLST Variables (Continued)

Variable	Description	Example
typeService	TypeServiceMBean MBean.	wls:/mydomain/serverConfig>**mi=typeService.getMBeanInfo('weblogic.management.configuration.ServerMBean')**
username	Name of user currently connected to WLST.	wls:/mydomain/serverConfig>**print username** weblogic
version	Current version of the running server to which WLST is connected.	wls:/mydomain/serverConfig>**print version** WebLogic Server 9.0 Thu Aug 31 12:15:50 PST 2005 778899

WLST Online and Offline Command Summary

The following sections summarize the WLST commands, as follows:

Note: You can list a summary of all online and offline commands from the command-line using the following commands, respectively:

```
help("online")
help("offline")
```

WLST Command Summary, Alphabetically By Command

The following tables summarizes each of the WLST commands, alphabetically by command.

Table B-1 WLST Command Summary

This command...	Enables you to...	Use with WLST...
"activate" on page A-41	Activate changes saved during the current editing session but not yet deployed.	Online
"addListener" on page A-71	Add a JMX listener to the specified MBean.	Online

Table B-1 WLST Command Summary (Continued)

This command...	Enables you to...	Use with WLST...
"addTemplate" on page A-8	Extend the current domain using an application or service extension template.	Offline
"assign" on page A-42	Assign resources to one or more destinations.	Offline
"assignAll" on page A-45	Assign all applications or services to one or more destinations. **Note:** This command is deprecated as of WebLogic Server 9.0. You should update your scripts to use the `assign` command, as described in "assign" on page A-42.	Offline
"cancelEdit" on page A-46	Cancel an edit session, release the edit lock, and discard all unsaved changes. This operation can be called by any user with administrator privileges, even if the user did not start the edit session.	Online
"cd" on page A-3	Navigate the hierarchy of configuration or runtime beans.	Online or Offline
"closeDomain" on page A-9	Close the current domain.	Offline
"closeTemplate" on page A-10	Close the current domain template.	Offline
"config" on page A-118	Navigate to the last MBean to which you navigated in the Administration or local configuration MBean hierarchy or to the root of the hierarchy, `DomainMBean`. **Note:** This command is deprecated as of WebLogic Server 9.0. You should update your script to use the `serverConfig` command as described in "serverConfig" on page A-125.	Online
"configToScript" on page A-72	Convert an existing server configuration (`config` directory) to an executable WLST script.	Online or Offline
"connect" on page A-10	Connect WLST to a WebLogic Server instance.	Online or Offline
"create" on page A-47	Create a configuration bean of the specified type for the current bean.	Online or Offline

Table B-1 WLST Command Summary (Continued)

This command...	Enables you to...	Use with WLST...
"currentTree" on page A-4	Return the current location in the hierarchy.	Online
"custom" on page A-119	Navigate to the root of custom MBeans that are registered in the server.	Online
"delete" on page A-49	Delete an instance of a configuration bean of the specified type for the current configuration bean.	Online or Offline
"deploy" on page A-23	Deploy an application to a WebLogic Server instance.	Online
"disconnect" on page A-15	Disconnect WLST from a WebLogic Server instance.	Online
"distributeApplication" on page A-27	Copy the deployment bundle to the specified targets.	Online
"domainConfig" on page A-120	Navigate to the last MBean to which you navigated in the domain configuration hierarchy or to the root of the hierarchy, `DomainMBean`.	Online
"domainRuntime" on page A-121	Navigate to the last MBean to which you navigated in the domain runtime hierarchy or to the root of the hierarchy, `DomainRuntimeMBean`.	Online
"dumpStack" on page A-74	Display stack trace from the last exception that occurred while performing a WLST action, and reset the stack trace.	Online or Offline
"dumpVariables" on page A-74	Display all variables used by WLST, including their name and value.	Online or Offline
"edit" on page A-123	Navigate to the last MBean to which you navigated in the configuration edit MBean hierarchy or to the root of the hierarchy, `DomainMBean`.	Online
"encrypt" on page A-50	Encrypt the specified string.	Online
"exit" on page A-15	Exit WLST from the user session and close the scripting shell.	Online or Offline
"exportDiagnosticData" on page A-36	Execute a query against the specified log file.	Offline

Table B-1 WLST Command Summary (Continued)

This command...	Enables you to...	Use with WLST...
"exportDiagnosticDataFromServer" on page A-38	Executes a query on the server side and retrieves the exported WebLogic Diagnostic Framework (WLDF) data.	Online
"find" on page A-75	Find MBeans and attributes in the current hierarchy.	Online
"get" on page A-51	Return the value of the specified attribute.	Online or Offline
"getActivationTask" on page A-52	Return the latest `ActivationTask` MBean on which a user can get status.	Online
"getConfigManager" on page A-77	Return the latest `ConfigurationManagerBean` MBean which manages the change process.	Online
"getMBean" on page A-77	Return the MBean by browsing to the specified path.	Online
"getMBI" on page A-78	Return the `MBeanInfo` for the specified `MBeanType` or the `cmo` variable.	Online
"getPath" on page A-79	Return the MBean path for the specified MBean instance.	Online
"getWLDM" on page A-28	Return the WebLogic `DeploymentManager` object.	Online
"invoke" on page A-53	Invoke a management operation on the current configuration bean.	Online
"isRestartRequired" on page A-54	Determine whether a server restart is required.	Online
"jndi" on page A-124	Navigates to the JNDI tree for the server to which WLST is currently connected.	Online
"listChildTypes" on page A-79	List all the children MBeans that can be created or deleted for the `cmo`.	Online
"loadApplication" on page A-29	Load an application and deployment plan into memory.	Online or Offline
"loadDB" on page A-55	Load SQL files into a database.	Offline
"loadProperties" on page A-56	Load property values from a file.	Online and Offline

Table B-1 WLST Command Summary (Continued)

This command...	Enables you to...	Use with WLST...
"lookup" on page A-80	Look up the specified MBean.	Online
"ls" on page A-81	List all child beans and/or attributes for the current configuration or runtime bean.	Online or Offline
"man" on page A-84	Display help from `MBeanInfo` for the current MBean or its specified attribute.	Online
"migrate" on page A-95	Migrate services to a target server within a cluster.	Online
"nm" on page A-105	Determine whether WLST is connected to Node Manager.	Online
"nmConnect" on page A-106	Connect WLST to Node Manager to establish a session.	Online
"nmDisconnect" on page A-108	Disconnect WLST from a Node Manager session.	Online
"nmEnroll" on page A-108	Enroll the machine on which WLST is currently running.	Online
"nmGenBootStartupProps" on page A-110	Generates the Node Manager property files, `boot.properties` and `startup.properties`, for the specified server.	Online
"nmKill" on page A-111	Kill the specified server instance that was started with Node Manager.	Online
"nmLog" on page A-111	Return the Node Manager log.	Online
"nmServerLog" on page A-112	Return the server output log of the server that was started with Node Manager.	Online
"nmServerStatus" on page A-113	Return the status of the server that was started with Node Manager.	Online
"nmStart" on page A-114	Start a server in the current domain using Node Manager.	Online
"nmVersion" on page A-115	Return the Node Manager server version.	Online
"prompt" on page A-5	Toggle the display of path information at the prompt.	Online or Offline
"pwd" on page A-6	Display the current location in the configuration or runtime bean hierarchy.	Online or Offline

Table B-1 WLST Command Summary (Continued)

This command...	Enables you to...	Use with WLST...
"readDomain" on page A-17	Open an existing domain for updating.	Offline
"readTemplate" on page A-18	Open an existing domain template for domain creation.	Offline
"redeploy" on page A-30	Reload classes and redeploy a previously deployed application.	Online
"redirect" on page A-85	Redirect WLST output to the specified filename.	Online or Offline
"removeListener" on page A-86	Remove a listener that was previously defined.	Online
"resume" on page A-97	Resume a server instance that is suspended or in `ADMIN` state.	Online
"runtime" on page A-124	Navigate to the last MBean to which you navigated in the Runtime hierarchy or the root of all runtime objects, `DomainRuntimeMBean`. **Note:** This command is deprecated as of WebLogic Server 9.0. You should update your scripts to use the `serverRuntime` command, as described in "serverRuntime" on page A-126.	Online
"save" on page A-56	Save the edits that have been made but have not yet been saved.	Online
"serverConfig" on page A-125	Navigate to the last MBean to which you navigated in the configuration MBean hierarchy or to the root of the hierarchy, `DomainMBean`.	Online
"serverRuntime" on page A-126	Navigate to the last MBean to which you navigated in the runtime MBean hierarchy or to the root of the hierarchy, `ServerRuntimeMBean`.	Online
"set" on page A-57	Set the specified attribute value for the current configuration bean.	Online or Offline
"setOption" on page A-58	Set options related to a domain creation or update	Offline
"showChanges" on page A-61	Show the changes made by the current user during the current edit session.	Online

Table B-1 WLST Command Summary (Continued)

This command...	Enables you to...	Use with WLST...
"showListeners" on page A-87	Show all listeners that are currently defined.	Online
"shutdown" on page A-97	Gracefully shut down a running server instance or cluster.	Online
"start" on page A-100	Start a Managed Server instance or a cluster using Node Manager.	Online
"startApplication" on page A-31	Start an application, making it available to users.	Online
"startEdit" on page A-62	Start a configuration edit session on behalf of the currently connected user.	Online
"startNodeManager" on page A-115	Start Node Manager at default port (5556).	Online or Offline
"startRecording" on page A-87	Record all user interactions with WLST; useful for capturing commands to replay.	Online or Offline
"startServer" on page A-101	Start the Administration Server.	Online or Offline
"state" on page A-88	Returns a map of servers or clusters and their state using Node Manager.	Online
"stopApplication" on page A-33	Stop an application, making it un available to users.	Online
"stopEdit" on page A-63	Stop the current edit session, release the edit lock, and discard unsaved changes.	Online
"stopRecording" on page A-89	Stop recording WLST commands.	Online or Offline
"stopRedirect" on page A-90	Stop the redirection of WLST output to a file.	Online or Offline
"storeUserConfig" on page A-90	Create a user configuration file and an associated key file.	Online
"suspend" on page A-103	Suspend a running server.	Online
"threadDump" on page A-92	Display a thread dump for the specified server.	Online or Offline

Table B-1 WLST Command Summary (Continued)

This command...	Enables you to...	Use with WLST...
"undeploy" on page A-34	Undeploy an application from the specified servers.	Online
"updateApplication" on page A-35	Update an application configuration using a new deployment plan.	Online
"updateDomain" on page A-19	Update and save the current domain.	Offline
"unassign" on page A-64	Unassign applications or services from one or more destinations.	Offline
"unassignAll" on page A-66	Unassign all applications or services from one or more destinations. **Note:** This command is deprecated as of WebLogic Server 9.0. You should update your scripts to use the `unassign` command, as described in "unassign" on page A-64.	Offline
"undo" on page A-67	Revert all unsaved or unactivated edits.	Online
"validate" on page A-68	Validate the changes that have been made but have not yet been saved.	Online
"viewMBean" on page A-93	Display information about an MBean, such as the attribute names and values, and operations.	Online
"writeDomain" on page A-19	Write the domain configuration information to the specified directory.	Offline
"writeIniFile" on page A-94	Convert WLST definitions and method declarations to a Python (`.py`) file.	Online or Offline
"writeTemplate" on page A-21	Writes the domain configuration information to the specified domain template.	Offline

WLST Online Command Summary

The following table summarizes the WLST online commands, alphabetically by command.

Table B-2 WLST Online Command Summary

This command...	Enables you to...
"activate" on page A-41	Activate changes saved during the current editing session but not yet deployed.
"addListener" on page A-71	Add a JMX listener to the specified MBean.
"cancelEdit" on page A-46	Cancel an edit session, release the edit lock, and discard all unsaved changes. This operation can be called by any user with administrator privileges, even if the user did not start the edit session.
"cd" on page A-3	Navigate the hierarchy of configuration or runtime beans.
"config" on page A-118	Navigate to the last MBean to which you navigated in the configuration MBean hierarchy or to the root of all configuration beans, `DomainMBean`. **Note:** This command is deprecated as of WebLogic Server 9.0. You should update your script to use the `serverConfig` command as described in "serverConfig" on page A-125.
"configToScript" on page A-72	Convert an existing server configuration (`config` directory) to an executable WLST script.
"connect" on page A-10	Connect WLST to a WebLogic Server instance.
"create" on page A-47	Create a configuration bean of the specified type for the current bean.
"currentTree" on page A-4	Return the current tree location.
"custom" on page A-119	Navigate to the root of custom MBeans that are registered in the server.
"delete" on page A-49	Delete an instance of a configuration bean of the specified type for the current configuration bean.
"deploy" on page A-23	Deploy an application to a WebLogic Server instance.
"disconnect" on page A-15	Disconnect WLST from a WebLogic Server instance.
"distributeApplication" on page A-27	Copy the deployment bundle to the specified targets.
"domainConfig" on page A-120	Navigate to the last MBean to which you navigated in the domain configuration hierarchy or to the root of the hierarchy, `DomainMBean`.

Table B-2 WLST Online Command Summary (Continued)

This command...	Enables you to...
"domainRuntime" on page A-121	Navigate to the last MBean to which you navigated in the domain runtime hierarchy or to the root of the hierarchy, `DomainRuntimeMBean`.
"dumpStack" on page A-74	Display stack trace from the last exception that occurred, and reset the trace.
"dumpVariables" on page A-74	Display all variables used by WLST, including their name and value.
"edit" on page A-123	Navigate to the last MBean to which you navigated in the configuration edit MBean hierarchy or to the root of the hierarchy, `DomainMBean`.
"encrypt" on page A-50	Encrypt the specified string.
"exit" on page A-15	Exit WLST from the interactive session and close the scripting shell.
"exportDiagnosticDataFromServer" on page A-38	Execute a query on the server side and retrieves the exported WebLogic Diagnostic Framework (WLDF) data.
"find" on page A-75	Find MBeans and attributes in the current hierarchy.
"get" on page A-51	Return the value of the specified attribute.
"getActivationTask" on page A-52	Return the latest `ActivationTask` MBean on which a user can get status.
"getConfigManager" on page A-77	Return the latest `ConfigurationManagerBean` MBean which manages the change process.
"getMBean" on page A-77	Return the MBean by browsing to the specified path.
"getMBI" on page A-78	Return the `MBeanInfo` for the specified `MBeanType` or the `cmo` variable.
"getPath" on page A-79	Return the MBean path for the specified MBean instance.
"getWLDM" on page A-28	Return the WebLogic `DeploymentManager` object.
"invoke" on page A-53	Invoke a management operation on the current configuration bean.
"isRestartRequired" on page A-54	Determine whether a server restart is required.
"jndi" on page A-124	Navigates to the JNDI tree for the server to which WLST is currently connected.

Table B-2 WLST Online Command Summary (Continued)

This command...	Enables you to...
"listChildTypes" on page A-79	List all the children MBeans that can be created or deleted for the `cmo`.
"loadApplication" on page A-29	Load an application and deployment plan into memory.
"loadProperties" on page A-56	Load property values from a file.
"lookup" on page A-80	Look up the specified MBean.
"ls" on page A-81	List all child beans and/or attributes for the current configuration or runtime bean.
"man" on page A-84	Display help from `MBeanInfo` for the current MBean or its specified attribute.
"migrate" on page A-95	Migrate services to a target server within a cluster.
"nm" on page A-105	Determine whether WLST is connected to Node Manager.
"nmConnect" on page A-106	Connect WLST to Node Manager to establish a session.
"nmDisconnect" on page A-108	Disconnect WLST from a Node Manager session.
"nmEnroll" on page A-108	Enroll the machine on which WLST is currently running.
"nmGenBootStartupProps" on page A-110	Generates the Node Manager property files, `boot.properties` and `startup.properties`, for the specified server.
"nmKill" on page A-111	Kill the specified server instance that was started with Node Manager.
"nmLog" on page A-111	Return the Node Manager log.
"nmServerLog" on page A-112	Return the server output log of the server that was started with Node Manager.
"nmServerStatus" on page A-113	Return the status of the server that was started with Node Manager.
"nmStart" on page A-114	Start a server in the current domain using Node Manager.
"nmVersion" on page A-115	Return the Node Manager server version.
"prompt" on page A-5	Toggle the display of path information at the prompt.
"pwd" on page A-6	Display the current location in the configuration or runtime bean hierarchy.

Table B-2 WLST Online Command Summary (Continued)

This command...	Enables you to...
"redeploy" on page A-30	Reload classes and redeploy a previously deployed application.
"redirect" on page A-85	Redirect WLST output to the specified filename.
"removeListener" on page A-86	Remove a listener that was previously defined.
"resume" on page A-97	Resume a server instance that is suspended or in `ADMIN` state.
"runtime" on page A-124	Navigate to the last MBean to which you navigated in the Runtime hierarchy or the root of all runtime objects, `DomainRuntimeMBean`. **Note:** This command is deprecated as of WebLogic Server 9.0. You should update your scripts to use the `serverRuntime` command, as described in "serverRuntime" on page A-126.
"save" on page A-56	Save the edits that have been made but have not yet been saved.
"serverConfig" on page A-125	Navigate to the last MBean to which you navigated in the configuration MBean hierarchy or to the root of the hierarchy, `DomainMBean`.
"serverRuntime" on page A-126	Navigate to the last MBean to which you navigated in the runtime MBean hierarchy or to the root of the hierarchy, `ServerRuntimeMBean`.
"set" on page A-57	Set the specified attribute value for the current configuration bean.
"showChanges" on page A-61	Show the changes made by the current user during the current edit session.
"showListeners" on page A-87	Show all listeners that are currently defined.
"shutdown" on page A-97	Gracefully shut down a running server instance or cluster.
"start" on page A-100	Start a Managed Server instance or a cluster using Node Manager.
"startApplication" on page A-31	Start an application, making it available to users.
"startEdit" on page A-62	Start a configuration edit session on behalf of the currently connected user.
"startNodeManager" on page A-115	Start Node Manager at default port (5556).
"startRecording" on page A-87	Record all user interactions with WLST; useful for capturing commands to replay.

Table B-2 WLST Online Command Summary (Continued)

This command...	Enables you to...
"startServer" on page A-101	Start the Administration Server.
"state" on page A-88	Returns a map of servers or clusters and their state using Node Manager
"stopApplication" on page A-33	Stop an application, making it un available to users.
"stopEdit" on page A-63	Stop the current edit session, release the edit lock, and discard unsaved changes.
"stopRecording" on page A-89	Stop recording WLST commands.
"stopRedirect" on page A-90	Stop the redirection of WLST output to a file.
"storeUserConfig" on page A-90	Create a user configuration file and an associated key file.
"suspend" on page A-103	Suspend a running server.
"threadDump" on page A-92	Display a thread dump for the specified server.
"undeploy" on page A-34	Undeploy an application from the specified servers.
"undo" on page A-67	Revert all unsaved or unactivated edits.
"updateApplication" on page A-35	Update an application configuration using a new deployment plan.
"validate" on page A-68	Validate the changes that have been made but have not yet been saved.
"viewMBean" on page A-93	Display information about an MBean, such as the attribute names and values, and operations.
"writeIniFile" on page A-94	Convert WLST definitions and method declarations to a Python (`.py`) file.

WLST Offline Command Summary

The following table summarizes the WLST offline commands, alphabetically by command.

Table B-3 WLST Offline Command Summary

This command...	Enables you to...
"addTemplate" on page A-8	Extend the current domain using an application or service extension template.
"assign" on page A-42	Assign resources to one or more destinations.
"assignAll" on page A-45	Assign all applications or services to one or more destinations. **Note:** This command is deprecated as of WebLogic Server 9.0. You should update your scripts to use the `assign` command, as described in "assign" on page A-42.
"cd" on page A-3	Navigate the hierarchy of configuration or runtime beans.
"closeDomain" on page A-9	Close the current domain.
"closeTemplate" on page A-10	Close the current domain template.
"configToScript" on page A-72	Convert an existing server configuration (`config` directory) to an executable WLST script.
"connect" on page A-10	Connect WLST to a WebLogic Server instance.
"create" on page A-47	Create a configuration bean of the specified type for the current bean.
"delete" on page A-49	Delete an instance of a configuration bean of the specified type for the current configuration bean.
"dumpStack" on page A-74	Display stack trace from the last exception that occurred while performing a WLST action, and reset the stack trace.
"dumpVariables" on page A-74	Display all variables used by WLST, including their name and value.
"exit" on page A-15	Exit WLST from the interactive session and close the scripting shell.
"exportDiagnosticData" on page A-36	Execute a query against the specified log file.
"get" on page A-51	Return the value of the specified attribute.
"loadDB" on page A-55	Load SQL files into a database.
"loadProperties" on page A-56	Load property values from a file.

Table B-3 WLST Offline Command Summary (Continued)

This command...	Enables you to...
"ls" on page A-81	List all child beans and/or attributes for the current configuration or runtime bean.
"nmConnect" on page A-106	Connect WLST to Node Manager to establish a session.
"prompt" on page A-5	Toggle the display of path information at the prompt.
"pwd" on page A-6	Display the current location in the configuration or runtime bean hierarchy.
"readDomain" on page A-17	Open an existing domain for updating.
"readTemplate" on page A-18	Open an existing domain template for domain creation.
"redirect" on page A-85	Redirect WLST output to the specified filename.
"set" on page A-57	Set the specified attribute value for the current configuration bean.
"setOption" on page A-58	Set options related to a domain creation or update.
"startNodeManager" on page A-115	Start Node Manager at default port (5556).
"startRecording" on page A-87	Record all user interactions with WLST; useful for capturing commands to replay.
"startServer" on page A-101	Start the Administration Server.
"stopRecording" on page A-89	Stop recording WLST commands.
"stopRedirect" on page A-90	Stop the redirection of WLST output to a file.
"threadDump" on page A-92	Display a thread dump for the specified server.
"unassign" on page A-64	Unassign applications or services from one or more destinations.
"unassignAll" on page A-66	Unassign all applications or services from one or more destinations. **Note:** This command is deprecated as of WebLogic Server 9.0. You should update your scripts to use the `unassign` command, as described in "unassign" on page A-64.
"updateDomain" on page A-19	Update and save the current domain.

Table B-3 WLST Offline Command Summary (Continued)

This command...	Enables you to...
"writeDomain" on page A-19	Write the domain configuration information to the specified directory.
"writeIniFile" on page A-94	Convert WLST definitions and method declarations to a Python (`.py`) file.
"writeTemplate" on page A-21	Writes the domain configuration information to the specified domain template.

WLST Deployment Objects

The following sections describe the WLST deployment objects:

WLSTPlan Object

The `WLSTPlan` object enables you to make changes to an application deployment plan after loading an application using the `loadApplication` command, as described in "loadApplication" on page A-29.

The following table describes the `WLSTPlan` object methods that you can use to operate on the deployment plan.

Table C-1 WLSTPlan Object Methods

To operate on the...	Use this method...	To...
Deployment Plan	`DeploymentPlanBean getDeploymentPlan()`	Return the `DeploymentPlanBean` for the current application.
	`void save() throws FileNotFoundException, ConfigurationException`	Saves the deployment plan to a file from which it was read.

Table C-1 WLSTPlan Object Methods (Continued)

To operate on the...	Use this method...	To...
Module Overrides	`ModuleOverrideBean createModuleDescriptor(String name, String uri, String moduleOverrideName)`	Create a `ModuleDescriptorBean` with the specified *`name`* and *`uri`* for the `ModuleOverrideBean` *`moduleOverrideName`*
	`ModuleOverrideBean createModuleOverride(String name, String type)`	Create a `ModuleOverrideBean` with the specified *`name`* and *`type`* for the current deployment plan.
	`void destroyModuleOverride(String name)`	Destroy the `ModuleOverrideBean` *`name`* in the deployment plan.
	`ModuleOverrideBean[] getModuleOverride(String name)`	Return the `ModuleOverrideBean` *`name`*.
	`ModuleOverrideBean[] getModuleOverrides()`	Return all `ModuleOverrideBean` objects that are available in the deployment plan.
	`VariableBean[] setModuleOverride(ModuleOverrideBean moduleOverride)`	Set the `ModuleOverrideBean` *`moduleOverride`* for the current deployment plan.
	`void showModuleOverrides()`	Prints all of the `ModuleOverrideBean` objects that are available in the deployment plan as name/type pairs.

Table C-1 WLSTPlan Object Methods (Continued)

To operate on the...	Use this method...	To...
Variables	`VariableBean createVariable(String name)`	Create a `VariableBean` *name* that can override values in the deployment plan.
	`void destroyVariable(String name)`	Destroy the `VariableBean` *name*.
	`VariableBean getVariable(String name)`	Return the `VariableBean` *name*.
	`VariableBean[] getVariables()`	Return all `VariableBean` objects that are available in the deployment plan.
	`void setVariable(String name, String value)`	Set the variable *name* to the specified *value*.
	`void setVariableBean(VariableBean bean)`	Set the `VariableBean` *bean*.
	`void showVariables()`	Print all of the `VariableBean` objects in the deployment plan as name/value pairs.

Table C-1 WLSTPlan Object Methods (Continued)

To operate on the...	Use this method...	To...
Variable Assignment	`VariableAssignmentBean createVariableAssignment(String name, String moduleOverrideName, String moduleDescriptorName)`	Create a `VariableAssignmentBean` for the `ModuleDescriptorBean` *`moduleDescriptorName`* for the `ModuleOverrideBean` *`moduelOverrideName`*.
	`void destroyVariableAssignment(String name, String moduleDescriptorName)`	Destroy the `VariableAssignmentBean` *`name`* for the `ModuleDescriptorBean` *`moduleDescriptorName`*.
	`VariableAssignmentBean getVariableAssignment(String name, String moduleDescriptorName)`	Return the `VariableAssignmentBean` *`name`* for the `ModuleDescriptorBean` *`moduleDescriptorName`*.
	`void showVariables()`	Prints all of the `VariableBean` objects in the deployment plan as name/value pairs.

WLSTProgress Object

The `WLSTProgress` object enables you to check the status of an executed deployment command. The `WLSTProgress` object is returned by the following commands:

- “deploy” on page A-23
- “distributeApplication” on page A-27
- “redeploy” on page A-30
- “startApplication” on page A-31
- “stopApplication” on page A-33
- “updateApplication” on page A-35

The following table describes the `WLSTProgress` object methods that you can use to check the status of the current deployment action.

Table C-2 WLSTProgress Object Methods

Use this method...	To...
`String getCommandType()`	Return the deployment `CommandType` of this event.This command returns one of the following values: `distribute`, `redeploy`, `start`, `stop`, or `undeploy`.
`String getMessage()`	Return information about the status of this event.
`ProgressObject getProgressObject()`	Return the `ProgressObject` that is associated with the current deployment action.
`String getState()`	Retrieve the state of the current deployment action. `CommandType` of this event.This command returns one of the following values: `running`, `completed`, `failed`, or `released` (indicating that the object has been released into production).
`boolean isCompleted()`	Determine if the current deployment action has been completed.
`boolean isFailed()`	Determine if the current deployment action has failed.
`boolean isRunning()`	Determine if the current deployment action is running.
`void printStatus()`	Print the current status of the deployment action, including the command type, the state, additional messages, and so on.

APPENDIX D

FAQs: WLST

General WLST

- On which versions of WebLogic Server is WLST supported?
- What is the relationship between WLST and the existing WebLogic Server command-line utilities, such as wlconfig and weblogic.Deployer?
- When would I choose to use WLST over the other command-line utilities or the Administration Console?
- What is the distinction between WLST online and offline?
- Is there a GUI that displays the MBeans in a Swing format, similar to wlshell?

Jython Support

- What version of Jython is used by WLST?
- Can I run regular Jython scripts from within WLST?

Using WLST

- If I have SSL or the administration port enabled for my server, how do I connect using WLST?
- In the event of an error, can you control whether WLST continues or exits?
- Why do I have to specify (and) after each command, and enclose arguments in single- or double-quotes?

Q. On which versions of WebLogic Server is WLST supported?

A. WLST online is supported on WebLogic Server 9.*x*, 8.1, and 7.0. WLST offline is supported on WebLogic Server 9.*x* and 8.1 SP5.

Q. What is the relationship between WLST and the existing WebLogic Server command-line utilities, such as wlconfig and weblogic.Deployer?

A. WLST functionality includes the capabilities of the following WebLogic Server command-line utilities:

- `weblogic.Admin` utility that you use to interrogate MBeans and configure a WebLogic Server instance (deprecated in this release of WebLogic Server)
- `wlconfig` Ant task tool for making WebLogic Server configuration changes (see "Using Ant Tasks to Configure and Use a WebLogic Server Domain" in *Developing Applications with WebLogic Server*)
- `weblogic.Deployer` utility for deploying applications. (see "Overview of Deployment Tools" in *Deploying Applications to WebLogic Server*)

Q. When would I choose to use WLST over the other command-line utilities or the Administration Console?

A. You can create, configure, and manage domains using WLST, command-line utilities, and the Administration Console interchangeably. The method that you choose depends on whether you prefer using a graphical or command-line interface, and whether you can automate your tasks by using a script.

Q. What is the distinction between WLST online and offline?

A. You can use WLST **online** (connected to a running Administration Server or Managed Server instance) and **offline** (not connected to a running server).

WLST online is used when you are connected to a running server and provides simplified access to Managed Beans (MBeans), WebLogic Server Java objects that you manage through JMX. Online, WLST provides access to information that is persisted as part of the internal representation of the configuration.

WLST offline enables you to create a new domain or update an existing domain without connecting to a running WebLogic Server—supporting the same functionality as the Configuration Wizard. Offline, WLST only provides access to information that is persisted in the `config` directory.

Q. Is there a GUI that displays the MBeans in a Swing format, similar to wlshell?

A. No. This type of GUI interface in not available.

Q. What version of Jython is used by WLST?

A. The WLST scripting environment is based on the Java scripting interpreter, Jython 2.1.

Q. Can I run regular Jython scripts from within WLST?

A. Yes. WebLogic Server developers and administrators can extend the WebLogic scripting language to suit their environmental needs by following the Jython language syntax. For more information, see `http://www.jython.org`.

Q. If I have SSL or the administration port enabled for my server, how do I connect using WLST?

A. If you will be connecting to a WebLogic Server instance through an SSL listen port on a server that is using the demonstration SSL keys and certificates, invoke WLST using the following command:

```
java -Dweblogic.security.SSL.ignoreHostnameVerification=true
-Dweblogic.security.TrustKeyStore=DemoTrust weblogic.WLST
```

Otherwise, at a command prompt, enter the following command:

```
java weblogic.WLST
```

Q. In the event of an error, can you control whether WLST continues or exits?

A. Yes, using the `exitonerror` variable. Set this variable to `true` to specify that execution should exit when WLST encounters an error, or `false` to continue execution. This variable defaults to `true`. For more information, see "WLST Variable Reference" on page A-127.

Q. Why do I have to specify (and) after each command, and enclose arguments in single- or double-quotes?

A. This is the proper Jython syntax. For more information, see `http://www.jython.org`.

Q. Can I start a server, deploy applications, and then shutdown the server using WLST?

A. Yes. For information about:

- Starting and shutting down servers, see "Life Cycle Commands" on page A-94.
- Deploying applications, see "Deployment Commands" on page A-22

Q. Can WLST connect to a Managed Server?

A. Yes. You can start and connect to a Managed Server using the `start` command and `connect` command, respectively. For more information, see "start" on page A-100 and "connect" on page A-10, respectively.

Q. Parameterization enables you to easily move configuration files between environments. For example, you may want to parameterize the log file locations. Does WLST support this type of parameterization?

A. Yes. You can use the `loadProperties` command to load your variables and values from a properties file. When you use the variables in your script, during execution, the variables are replaced with the actual values from the properties file.

Q. Does the `configToScript` command convert security MBeans in config.xml?

A. Yes, the security MBeans are converted. However, the information within the Embedded LDAP is not converted.

Q. How can I access custom MBeans that are registered in the WebLogic MBeanServer?

A. Navigate to the custom tree using the `custom` command. For more information, see "Tree Commands" on page A-117.

Q. Why am I not seeing all MBeans that are registered in the MBeanServer?

A. There are internal and undocumented MBeans that are not shown by WLST.

Also, because WLST offline enables you to access and update the configuration objects that appear in the configuration files only, if you wish to view and/or change attribute values for a configuration object that is not already persisted in the configuration files as an XML element, you must first create the configuration object.

Q. When browsing custom MBeans, why do I get the following error message: **`No stub Available`**?

A. When browsing the custom MBeans, the `cmo` variable is not available.

Q. Can I connect to a WebLogic Server instance via HTTP?

A. If you are connecting to a WebLogic Server instance via HTTP, ensure that the `TunnelingEnabled` attribute is set to `true` for the WebLogic Server instance. For more information, see "TunnelingEnabled" in *WebLogic Server MBean Reference*.

Q. Can I invoke WLST via Ant?

A. Yes, one could fork a new weblogic.WLST process inside an Ant script and pass your script file as an argument.

Q. Can WLST scripts execute on the server side?

A. Yes. You can instantiate an instance of the WLST interpreter in your Java code and use it to run WLST commands and scripts. You can then call the WLST scripts as a startup class or as part of `ejbCreate` so that they execute on the server side. For more information, see "Embedded Mode" on page 2-6.

Q. Can I customize WLST?

A. Yes. You can update the WLST home directory to define custom WLST commands, WLST commands within a library, and WLST commands as a Jython module. For more information, see "Customizing WLST" on page 2-18.