BEA WebLogic Server®

WebLogic Scripting Tool

Version 9.0 BETA
Revised: December 15, 2004

Contents

Introduction and Roadmap

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

- "Document Scope and Audience" on page 1-1
- "Guide to This Document" on page 1-1
- "Related Documentation" on page 1-2
- "WLST Sample Scripts" on page 1-2

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 navigate, interrogate, and edit MBeans using WLST while connected to a running WebLogic Server instance.
- 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 server 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.

Related Documentation

- "Using Ant Tasks to Configure a WebLogic Server Domain" in *Developing Applications With WebLogic Server.*
- *Administration Console Online Help*
- *Developing Manageable Applications with JMX*
- *Creating WebLogic Domains Using the Configuration Wizard*

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 2-9.

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 Platform installation, such as `c:\beahome\weblogic90\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`.
`JDBCPool_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 `JDBCPool_creation.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 Platform.

Table 1-2 summarizes WLST offline sample scripts.

Table 1-2 WLST Offline Sample Script

WLST Sample Script	Description

Table 1-2 WLST Offline Sample Script (Continued)

`domainMedrec.py`	Uses the Avitek Medical Records Sample template to show you two methods for creating distributed queues.
`domainWLS.py`	Creates a WebLogic domain using the Basic WebLogic Server Domain template. The script demonstrates how to open a domain template, create and edit configuration objects, and write the domain configuration information to the specified directory. This sample script uses WLST offline only.

Using the WebLogic Scripting Tool

The following sections describe the WebLogic Scripting Tool:

- "What is the WebLogic Scripting Tool?" on page 2-1
- "Modes of Operation" on page 2-3
- "Main Steps for Using WLST" on page 2-7
- "Getting Help" on page 2-11
- "Recording User Interactions" on page 2-12

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:

- 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 `config.xml` file. See "Editing Configuration MBeans" on page 4-11.
- 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 4-10.
- Automate domain configuration tasks and application deployment. See "Automating WebLogic Server Administration Tasks" on page 6-1.
- Clone WebLogic Server domains. See "Creating and Configuring WebLogic Domains Using WLST Offline" on page 3-1.
- Control and manage the server life cycle. See "Managing Servers and Server Life Cycle" on page 5-1.
- Access 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 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 WebLogic Server Applications*. Describes several tools that WebLogic Server provides for deploying applications and standalone 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), WebLogic Server Java objects that you 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 Manageable Applications 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 values that are stored in the domain `config.xml` file or in a domain template JAR created using Template Builder.

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.

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.

For more information, see "Recording User Interactions" on page 2-12.

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 2-9.

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 the example script, 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.

For information on how to run this script, see "Running Scripts" on page 2-9.

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” on page 2-8
- “Invoking WLST” on page 2-8
- “Requirements for Entering WLST Commands” on page 2-9
- “Running Scripts” on page 2-9
- “Importing WLST as a Jython Module” on page 2-10
- “Exiting WLST” on page 2-11

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 Platform.

   You can use a *`WL_HOME`*`\server\bin\setWLSenv` script to set both variables. See "Setting Up the Environment" on page 6-2.

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, you 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 and connect WLST to the server using the `connect` command.

   ```
   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'.
   ```

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

For information on starting WebLogic Server, see "Overview of Starting and Stopping Servers" in *Managing Server Startup and Shutdown*.

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"`.
- Specify configuration object pathnames using the forward slash character (`/`) as the delimiter on both Windows and UNIX, according to the Jython syntax. To specify a configuration object 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. For more information, see "Getting Help" on page 2-11.

Running Scripts

WLST incorporates two Jython functions that support running scripts: `java weblogic.WLST` *`filePath`*`.py`, which invokes WLST and executes a script file in a single command, and `execfile(`*`filePath`*`.py)` which executes a script file after you invoke WLST.

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 1-2.

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
```

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 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 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 prompt 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')
   ....
   ```

Exiting WLST

To exit WLST:

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

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.

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, the following command displays information about the `disconnect` 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)
stopRecording()
```

You must specify the file pathname for storing WLST commands when you enter the `startrecording` command.

For more information, see "startRecording" on page A-64 and "stopRecording" on page A-65.

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 values that are stored in the domain `config.xml` file 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:

- "Creating a Domain (Offline)" on page 3-2
- "Updating an Existing Domain (Offline)" on page 3-3
- "Stepping Through a Sample Script: Creating a Domain" on page 3-5

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

Note: 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 2-7.

You can exit WLST at anytime using the `exit` command, as follows: `exit()`

Creating a Domain (Offline)

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

Table 3-1 Steps for Creating a Domain (Offline)

Step	To…	Use this command…	For more information, see…
1	Open an existing domain template for domain creation	`readTemplate(`*`templateFileName`*`)`	"readTemplate" on page A-14
2	Modify the domain (optional)	Various commands	"Browsing and Accessing Information About the Configuration Bean Hierarchy (Offline)" on page 3-4 "Editing a Domain (Offline)" on page 3-5.
3	Write the domain configuration information to the specified directory	`writeDomain(`*`domainDir`*`)`	"writeDomain" on page A-15
4	Close the current domain template	`closeTemplate()`	"closeTemplate" on page A-9

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-13
2	Extend the current domain (optional)	`addTemplate(`*`templateFileName`*`)`	"addTemplate" on page A-8
3	Modify the domain (optional)	Various commands	"Browsing and Accessing Information About the Configuration Bean Hierarchy (Offline)" on page 3-4 "Editing a Domain (Offline)" on page 3-5.
4	Save the domain	`updateDomain()`	"updateDomain" on page A-14
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:

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-59
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-55
Display the stack trace from the last exception that occurred while performing a WLST action	`dumpStack()`	"dumpStack" on page A-55

Editing a Domain (Offline)

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

Table 3-4 Editing a Domain

To...	Use this command...	For more information, see...

Assign configuration beans	`assign(`*`sourceType, sourceName, destinationType, destinationName`*`)`
	`assignAll(`*`sourceType, destinationType, destinationName`*`)`
		"assign" on page A-32
		"assignAll" on page A-34

Unassign configuration beans	`unassign(`*`sourceType, sourceName, destinationType, destinationName`*`)`
	`unassignAll(`*`sourceType, destinationType, destinationName`*`)`
		"unassign" on page A-48
		"unassignAll" on page A-50

Create and delete configuration beans	`create(`*`name, childBeanType`*`)`
	`delete(`*`name, childBeanType`*`)`
		"create" on page A-35
		"delete" on page A-37

Get and set attribute values	`get(`*`attributeName`*`)`
	`set(`*`attributeName, attributeValue`*`)`
		"get" on page A-37
		"set" on page A-43

Set configuration options	`setOption(`*`optionName, optionValue`*`)`	"setOption" on page A-44
Load SQL files into a database	`loadDB(`*`dbVersion, connectionPoolName`*`)`	"loadDB" on page A-41

Stepping Through a Sample Script: Creating a Domain

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 sample script, `domain_wls.py`, is located at `c:\bea\weblogic90\samples\scripts\wlst`.

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

1. Open an existing domain template (assuming WebLogic Platform is installed at `c:/bea/weblogic90`). In this example, we open the Basic WebLogic Server Domain template.

   ```
   readTemplate(‘c:/bea/weblogic90/common/templates/domains/wls.jar’)
   ```

2. Configure the domain.

   a. Configure the Administration Server.

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

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

   b. Create a JMSQueue.

   ```
   cd('/')
   create('myJMSServer', 'JMSServer')
   cd('JMSServer/myJMSServer')
   myq=create('myJMSQueue','JMSQueue')

   myq.setJNDIName('jms/myjmsqueue')
   set myq.StoreEnabled('false')
   ```

   c. Create a JDBCConnectionPool.

   ```
   cd('/')
   mypool=create('demoPool', 'JDBCConnectionPool')
   mypool.setDriverName('com.pointbase.jdbc.jdbcUniversalDriver')
   mypool.setURL('jdbc:pointbase:server://localhost:9092/demo')
   mypool.setPassword('PBPUBLIC')
   assign('JDBCConnectionPool', '*', 'Target', 'BEA_Server')
   ```

   d. Create two servers and target them to a cluster.

   ```
   create('newServer1', 'Server')
   create('newServer2', 'Server')
   create('myCluster', 'Cluster')
   assign('Server', 'newServer1,newServer2','Cluster','myCluster')
   ```

   e. Target existing applications, if any exist in the current domain, to the cluster.

   ```
   assign('Applications', '*', 'Target' 'myCluster')
   ```

   f. Define the user password.

```
cd('/')
cd('Security/BEA_DOMAIN')

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

3. Save the domain.

```
setOption('OverwriteDomain', 'true')
writeDomain('c:/bea1/user_projects/domains/wls_testscript')
```

4. Close the current domain template.

```
closeTemplate()
```

5. Exit WLST.

```
exit()
```

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 MBeans exist within hierarchical structures. 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.

In the configuration hierarchy, the root directory is `DomainMBean`; 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.

For more information, see DomainMBean in *WebLogic Server MBean Reference*.

Figure 4-1 Configuration MBean Hierarchy

```
Domain MBean (root)
     |- - - MBean type (LogMBean)
                |- - - MBean instance (medrec)
                           |- - - MBean attributes & operations (FileName)
     |- - - MBean type (RealmMBean)
     |- - - 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-86.

For more information about MBean hierarchies, see "Configuration, Runtime, and Domain-Runtime Hierarchies" in *Developing Manageable Applications with JMX*.

Changing the Current Management Object

When WLST first connects to an instance of WebLogic Server through the interactive scripting shell, 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.

Listing 4-1 Changing the Current Management Object

```
C:\> java weblogic.WLST
Initializing WebLogic Scripting Tool (WLST) ...
Welcome to Weblogic Server Administration Scripting Shell
Type help() for help on available commands
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'.
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--   CachingRealms
dr--   Clusters
dr--   DeploymentConfiguration
dr--   Deployments
dr--   DomainLogFilters
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--   DefaultPersistentStore
dr--   ExecutiveQueues
dr--   IIOP
dr--   JTARecoveryService
dr--   KernelDebug
dr--   Log
dr--   NetworkAccessPoints
dr--   OverloadProtection
dr--   SSL
dr--   ServerDebug
dr--   ServerDiagnosticConfig
dr--   ServerRuntime
dr--   ServerStart
dr--   WebServer

-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()
-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
```

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, and `w` indicates a writeable property. (Table A-8 on page 59 describes the `ls` command output information.)

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 the Runtime Information

Similar to the configuration information, WebLogic Server Runtime MBeans are arranged in a hierarchical 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 DomainRuntime 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 *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:/(Not Connected) > 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--   HarvesterRuntime
```

```
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                              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'.
wls:/mydomain/serverConfig> serverRuntime()
wls:/mydomain/serverRuntime> ls()
dr--   ApplicationRuntimes
dr--   ClusterRuntime
dr--   HarvesterRuntime
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>
```

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.

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 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.

For more information, see “find” on page A-56 and “getMBean” on page A-39.

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 services, you can create and register your own MBeans within the MBean Server subsystem on the Administration Server.

For more information on Custom MBeans, see "Non-WebLogic Server MBeans" in *Programming WebLogic Management Services 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 4-11.

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, see "Configuration Management Process" in *Understanding Domain Configuration* and "Managing a Domain's Configuration with JMX" in *Developing Manageable Applications with JMX*.

Making Changes

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

1. To start the change process, enter the `edit` command.

   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.

   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.

   For detailed information about `startEdit` command arguments, see "startEdit" on page A-47.

   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.

   For more information, see "Edit Commands" on page A-29.

   Use the `validate` command to ensure that changes you make are valid before saving them.

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.

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. To undo a particular change, you must enter the `undo` command right after making the change.

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.

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.

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.

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

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()
```

By default, all configuration changes that you make using WLST are treated as a set of modifications, a batch change. WLST lets you override the default behavior of making batch changes by wrapping each command with `edit` and `save` functions; to do this, set the `autocommit` variable to `True`. See "WLST Variable Reference" on page A-95.

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-40.

Managing Changes

WebLogic Server provides a Configuration Manager service to manage the change process. The `getConfigurationManager` function returns the `ConfigurationManagerMBean`. You use `ConfigurationManagerMBean` methods to manage configuration changes across a domain.

For more information, see ConfigurationManagerMBean in the *Administration Console Online Help*.

The 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.

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 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.)

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 Configuration Manager activate task 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 more information, see ActivationTaskMBean in the *Administration Console Online Help*.

The 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.

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()
```

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 cycle through a number of different 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 stand-alone, command-line client to Node Manager 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" in *Managing Server Startup and Shutdown* and "Using Node Manager to Control Servers" in *Designing and Configuring WebLogic Server Environments*.

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.

See "Starting and Stopping Servers: Quick Reference" in *Managing Server Startup and Shutdown*.

Starting an Administration Server Without Node Manager

To start an Administration Server:

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. Run one of the following scripts:
   - `setDomainEnv.cmd` (Windows)
   - `setDomainENV.sh` (UNIX)
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')
   ```

For example,

```
startServer(‘myserver','mydomain','t3://localhost:7001','weblogic','web
logic','c://diablodomain')
```

   For detailed information about `startServer` command arguments, see “startServer” on page A-75.

This command starts the Administration Server stand-alone, without using Node Manager. The server runs in a separate process from WLST; exiting WLST does not shut down the server.

If you use the `startServer` command with WLST connected to Node Manager, you can use Node Manager to support starting, stopping, and monitoring the Administration Server.

See “Using WLST and Node Manager to Manage Servers” on page 5-4.

Starting a Managed Server With Node Manager

To start a Managed Server:

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 manually at a command prompt or with one of the following scripts:

   – `startNodeManager.cmd (Windows)`

   – `startNodeManager.sh (UNIX)`

   Sample start scripts for Node Manager are installed in the `WL_HOME\server\bin` directory, where `WL_HOME` is the top-level installation directory for WebLogic Platform.

   For more information, see “Starting and Stopping Node Manager” in *Designing and Configuring WebLogic Server Environments*.

   The WebLogic Server installation process installs Node Manager as a Windows service on Windows systems, so that it starts automatically when you boot a computer. BEA Systems recommends running Node Manager as an operating system service so that it is automatically restarted in the event of system failure or reboot and using Node Manager to start or restart servers.

   For more information, see “Installing the Node Manager as a Windows Service” in the *Installation Guide.*

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'.

   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','managedServerlistenAddress',managed
   ServerlistenPort)
   ```

For example,

```
start('managed1','Server','localhost',7701)
```

For detailed information about `start` command arguments, see "start" on page A-74.

This command starts Managed Servers or clusters in a domain using Node Manager and requires that Node Manager is running on the same machine on which WLST is running. To use the `start` command, WLST must be connected to a running Administration Server or to Node Manager.

Using WLST and Node Manager to Manage Servers

Node Manager is a utility for 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.

You can start, suspend, and stop server instances remotely or locally, using WLST as a Node Manager command-line 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 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. WLST must be connected to an Administration Server to run this command; WLST does not need to be connected to the Node Manager.

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 Managed Servers 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 manually at a command prompt or with one of the following scripts:

   – `startNodeManager.cmd` (Windows)

   – `startNodeManager.sh` (UNIX)

   Sample start scripts for Node Manager are installed in the *`WL_HOME`*`\server\bin` directory, where *`WL_HOME`* is the top-level installation directory for WebLogic Platform.

   For more information, see "Starting and Stopping Node Manager" in *Designing and Configuring WebLogic Server Environments*.

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

   ```
   wls:/offline>nmConnect('username','password','NMhost','NMport','domainN
   ame','domainDir','NMtype')
   ```

For example,

```
nmConnect("weblogic", "weblogic", "localhost", "5555", "BEA_DOMAIN",
"c:/bea90_29/user_projects/domains/BEA_DOMAIN","ssl")
```

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

For detailed information about `nmConnect` command arguments, see "nmConnect" on page A-79.

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

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

   ```
   wls:/nm/BEA_DOMAIN>nmStart('adminServerName')
   starting server BEA_Server
   ...
   Server BEA_Server started successfully
   wls:/nm/BEA_DOMAIN>
   ```

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

   ```
   wls:/nm/BEA_DOMAIN>nmServerStatus('adminServerName')
   RUNNING
   wls:/nm/BEA_DOMAIN>
   ```

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

   ```
   wls:/nm/BEA_DOMAIN>nmKill()
   Killing server BEA_Server
   Server BEA_Server killed successfully
   wls:/nm/BEA_DOMAIN>
   ```

For more information about WLST Node Manager commands, see "Node Manager Commands" on page A-77.

Monitoring Server State

WebLogic Server displays and stores information about the current 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-64.

- By navigating to the `ServerRuntimeMBean` and interrogating 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 6-10.

Managing Server State

WLST life cycle commands let you control the states through which a server instance transitions.

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

For information on how to run this script, see “Running Scripts” on page 2-9.

Listing 5-1 WLST Life Cycle Commands

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

# First enable the AdministrationPort
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")
```

Automating WebLogic Server Administration Tasks

You can use the WebLogic Scripting Tool (WLST) to automate the creation and management of domains, servers, and resources. WLST provides several commands that create, get and set values for, invoke operations on, and delete instances of Configuration MBeans. It also provides 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
- "Configuring Logging" on page 6-12

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—supporting 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 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 6-4
   - "Creating JMS Resources" on page 6-5
   - "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 Platform.

If you want to use JDBC resources to connect to a database, modify the environment as the database vendor requires. Usually this entails adding driver classes to the `CLASSPATH` variable and vendor-specific directories to the `PATH` variable. To set the environment that the sample

Pointbase database requires as well as add an SDK to `PATH` variable and the WebLogic Server classes to the `CLASSPATH` variable, invoke the following script:

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

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

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()
adminServerName="medrec-adminServer"
domainName="MedRecDomain"
url="t3://localhost:8001"
username="weblogic"
password="weblogic"
startServer(adminServerName,domainName,url,domainDir=domainDir.getPath(),block
="true")
connect(username, password, 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 7001. If the sample MedRec domain is running, the 7001 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

```
## Creating and Configuring a JDBC System Resource
jdbcSR = create("MedRecPool-Oracle","JDBCSystemResource")
theJDBCResource = jdbcSR.getJDBCResource()
theJDBCResource.setName("MedRecPool-Oracle")
connectionPoolParams = theJDBCResource.getJDBCConnectionPoolParams()
connectionPoolParams.setMaxCapacity(10)
connectionPoolParams.setInitialCapacity(4)
connectionPoolParams.setLoginDelaySeconds(1)
connectionPoolParams.setShrinkFrequencySeconds(900)
connectionPoolParams.setShrinkingEnabled(1)
connectionPoolParams.setTestConnectionsOnRelease(0)
connectionPoolParams.setTestTableName("dual")

driverParams = theJDBCResource.getJDBCDriverParams()
driverParams.setDriverName("oracle.jdbc.OracleDriver")
driverParams.setUrl("jdbc:oracle:thin:@baybridge:1531:bay920")
driverParams.setPassword("tiger")
driverProperties = driverParams.getProperties()

proper = driverProperties.createProperty("medrec")
proper.setName("user")
proper.setValue("scott")

# Creating a Transactional Data Source
jdbcTxSR = create("MedRecTxDataSource","JDBCSystemResource")
theJDBCTxResource = jdbcTxSR.getJDBCResource()
theJDBCTxResource.setName("MedRecTxDataSource")
theJDBCTxResource.setLegacyType(4)
dsTxParams = theJDBCTxResource.getJDBCDataSourceParams()
dsTxParams.setPoolName("MedRecPool-Oracle")
dsTxParams.setJndiName("MedRecTxDataSource")
```

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")
```

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 the Javadoc. For more information about the WebLogic Server mail service, see "Mail" 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/weblogic90/samples/server/medrec/src/physicianEa
r","medrec-adminServer",securityModel="Advanced",block="true")
deploy("StartupEAR","C:/bea/weblogic90/samples/server/medrec/src/startupEar","
medrec-adminServer",securityModel="Advanced",block="true")
deploy("MedRecEAR","C:/bea/weblogic90/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.
- 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.

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

1. Set the required environment variables.

   See "Setting Up the Environment" on page 6-2.

2. Invoke WLST and run the sample script in Listing 6-6 by entering the following command:

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

   For information on how to run this script, see "Running Scripts" on page 2-9.

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()
adminServerName="medrec-adminServer"
domainName="MedRecDomain"
url="t3://localhost:8001"
username="weblogic"
password="weblogic"
startServer(adminServerName,domainName,url,domainDir=domainDir.getPath(),block
="true")
connect(username, password, url)
edit()
startEdit()

# Creating and Configuring a JDBC System Resource
jdbcSR = create("MedRecPool-Oracle","JDBCSystemResource")
theJDBCResource = jdbcSR.getJDBCResource()
```

```
theJDBCResource.setName("MedRecPool-Oracle")
theJDBCResource.setLegacyType(1)
connectionPoolParams = theJDBCResource.getJDBCConnectionPoolParams()
connectionPoolParams.setMaxCapacity(10)
connectionPoolParams.setInitialCapacity(4)
connectionPoolParams.setLoginDelaySeconds(1)
connectionPoolParams.setShrinkFrequencySeconds(900)
connectionPoolParams.setShrinkingEnabled(1)
connectionPoolParams.setTestConnectionsOnRelease(0)
connectionPoolParams.setTestTableName("dual")

driverParams = theJDBCResource.getJDBCDriverParams()
driverParams.setDriverName("oracle.jdbc.OracleDriver")
driverParams.setUrl("jdbc:oracle:thin:@baybridge:1531:bay920")
driverParams.setPassword("tiger")
driverProperties = driverParams.getProperties()

proper = driverProperties.createProperty("medrec")
proper.setName("user")
proper.setValue("scott")

# Creating a Transactional Data Source
jdbcTxSR = create("MedRecTxDataSource","JDBCSystemResource")
theJDBCTxResource = jdbcTxSR.getJDBCResource()
theJDBCTxResource.setName("MedRecTxDataSource")
theJDBCTxResource.setLegacyType(4)
dsTxParams = theJDBCTxResource.getJDBCDataSourceParams()
dsTxParams.setPoolName("MedRecPool-Oracle")
dsTxParams.setJndiName("MedRecTxDataSource")

# 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.setTunnelingEnabled(1)
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)
jdbcTxSR.addTarget(tgt)
jmsSystemResource.addTarget(tgt)
mrjStore.addTarget(tgt)
mrJMSServer.addTarget(tgt)
mrMailSession.addTarget(tgt)

save()
activate(block="true")

# Deploying Applications

deploy("PhysicianEAR","C:/bea/weblogic90/samples/server/medrec/src/physicianEa
r","medrec-adminServer",securityModel="Advanced",block="true")

deploy("StartupEAR","C:/bea/weblogic90/samples/server/medrec/src/startupEar","
medrec-adminServer",securityModel="Advanced",block="true")

deploy("MedRecEAR","C:/bea/weblogic90/samples/server/medrec/src/medrecEar","me
drec-adminServer",securityModel="Advanced",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 `DomainRuntime` 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()
   ```

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

Script for Monitoring Server State

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

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 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.

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()
```

Configuring Logging

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

To determine which log attributes can be configured, view the Javadoc for the `LogMBean` and the Javadoc for the `LogFileMBean`. The Javadoc also indicates valid values for each attribute.

The commands in Listing 6-9 get and set several `LogMBean` and `LogFileMBean` attributes.

For information on how to run this script, see “Running Scripts” on page 2-9.

Listing 6-9 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\\adminServer\\admin.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)

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

print
ls ()
```

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

WLST Command and Variable Reference

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

Overview of WSLT Command Categories

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 Command	The `exportDiagnosticData` command exports the diagnostic data created by WLST.
Edit Commands	Interrogate and edit configuration beans.
Life Cycle Commands	Manage the life cycle of a server instance.
Information Commands	Interrogate domains, servers, and variables, and provide configuration bean, runtime bean, and WLST-related information.
Node Manager Commands	Start, shut down, restart, and monitor remote WebLogic Server instances using Node Manager.
Tree Commands	Navigate the hierarchy of MBeans.

See also "Requirements for Entering WLST Commands" on page 2-9.

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 bean, enter `cd('..')`. (The special file name, `..` (dot-dot), refers to the directory immediately above the current directory.) To get back to the root configuration bean after navigating to a configuration bean that is deep in the hierarchy, enter `cd('/')`.

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

The `cd` command returns a stub of the configuration bean instance, if one exists.

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 MBean type until you navigate to an MBean instance. For more information about the `cmo` variable, see "Changing the Current Management Object" on page 4-2.

Syntax

```
cd(path)
```

Argument	Definition
path	Path to the configuration 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.

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()
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 configuration bean navigation 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 bean 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.

Syntax

```
prompt(['off'|'on'])
```

Argument	Definition
`'off' | 'on'`	Optional. Hides or displays WLST prompt: • The `off` argument hides the WLST prompt and 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`. If you subsequently enter the `prompt()` command without arguments, WLST displays the WLST command prompt without the configuration bean navigation path information. To redisplay the configuration bean path information, enter `prompt()` again, or enter `prompt('on')`. • The `on` argument displays the default WLST prompt, including the configuration bean navigation path information.

Examples

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

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

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

The following example hides the prompt and defaults to the Jython prompt, 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()
'Servers/myserver'
wls:> prompt('on')
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 configuration bean navigation path using the `prompt` command, as described in "prompt" on page A-5.

Syntax

```
pwd()
```

Example

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

```
wls:/mydomain/serverConfig/Servers/myserver/Log/myserver> pwd()
'/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-12	Disconnect WLST from a WebLogic Server instance.	Online
Create a new domain from a domain template	"readTemplate" on page A-14	Open an existing domain template for domain creation.	Offline
	"writeDomain" on page A-15	Write the domain configuration information to the specified directory.	Offline
	"closeTemplate" on page A-9	Close the current domain template.	Offline

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

In order to...	Use this command...	To...	Use with WLST...
Update an existing domain (offline)	"readDomain" on page A-13	Open an existing domain for updating.	Offline
	"addTemplate" on page A-8	Extend the current domain using the specified application or service extension template.	Offline
	"updateDomain" on page A-14	Update and save the current domain.	Offline
	"closeDomain" on page A-9	Close the current domain.	Offline
Exit WLST	"exit" on page A-13	Disconnect 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 the specified application or service extension template.

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/weblogic90/common/templates/
applications/DefaultWebApp.jar')
```

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.

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()
```

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.

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()
```

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 use 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 more information, see "Specifying User Credentials" in "weblogic.Admin Command-Line Reference" in *WebLogic Server Command Reference*.

If you run this command from the domain directory in which the server was started, you do not need to specify the username and password. In this case, the user configuration file and key file are used.

Please note:

- 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 2-7.

- If 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 Configuration Reference*.

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

Syntax

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

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

Argument	Definition
username	Username of the operator who is connecting WLST to the server. If not specified on the command line, WLST prompts for the username.
password	Password of the operator who is connecting WLST to the server. If not specified on the command line, WLST prompts you for the password.
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	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-66.)
userKeyFile	Name and location of the key file that is associated with the specified user configuration file and is used to decrypt it.

Examples

The following example connects WLST to a WebLogic Server instance.

```
wls:/offline> connect('system','weblogic')
Connecting to weblogic server instance running at t3://localhost:7001 as
username system...
Successfully connected to Admin Server 'myserver' that belongs to domain
'mydomain'.
wls:/mydomain/serverConfig>
```

The following example connects WLST to a WebLogic Server instance at the specified URL.

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

```
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 'myserver' that belongs to domain
'mydomain'.
wls:/mydomain/serverConfig>
```

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.

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

Syntax

```
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.

Syntax

```
exit()
```

Example

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

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

readDomain

Command Category: Control Commands
Use with WLST: Offline

Description

Opens an existing domain for updating.

Syntax

```
readDomain(domainDirName)
```

Argument	Definition
domainDirName	Specifies the directory name of the domain 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.

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/weblogic90/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.

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. The domain continues to be editable after you execute this 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`.

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 do.ain configuration information to the `c:/bea/user_projects/domains/medrec` directory.

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

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 WebLogic Server Applications*.

Table A-4 Deployment Commands for WLST Configuration

This command...	Enables you to...	Use with WLST...
"deploy" on page A-17	Deploy an application.	Online
"distributeApplication" on page A-19	Copies the complete deployment bundle, module, configuration data, and any additional generated code to the specified targets.	Online
"getWLDM" on page A-21	Returns the WebLogic Deployment Manager object.	Online
"loadApplication" on page A-21	Loads an application and deployment plan into memory.	Online
"redeploy" on page A-22	Redeploy a previously deployed application	Online
"startApplication" on page A-24	Starts an application, making it available to users.	Online
"stopApplication" on page A-25	Stops an application, making it unavailable to users.	Online

Table A-4 Deployment Commands for WLST Configuration (Continued)

This command...	Enables you to...	Use with WLST...
"undeploy" on page A-26	Undeploy an application.	Online
"updateApplication" on page A-27	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.

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. A comma-separated list of the target servers. 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. The staging mode for the application you are deploying. Valid values are `stage`, `nostage`, and `external_stage`. For information about the staging modes, see "Staging Modes" in "Overview of WebLogic Server Deployment" in *Deploying WebLogic Server Applications*.This argument defaults to null.

Argument	Definition (Continued)
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. Valid options include: • **block**—Boolean. Specifies whether WLST should block user interaction until the command completes. This argument 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. • **clusterDeploymentTimeout**—Time, in milliseconds, granted for a cluster deployment task on this application. • **gracefulIgnoreSessions**—Boolean. Specifies whether the graceful production to admin mode operation should ignore pending HTTP sessions. This argument defaults to `false` and only applies if `gracefulProductionToAdmin` is set to `true`. • **gracefulProductionToAdmin**—Boolean. Specifies whether the production to admin mode operation should be graceful. This argument defaults to `false`. • **libraryModule**—Boolean. Specifies whether the module is a library module. This argument defaults to `false`. • **retireGracefully**—Specifies the 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 argument default to `-1`, which specifies graceful timeout. • **securityModel**—Defines the security model. Valid values include: `DDOnly`, `CustomRoles`, `CustomRolesAndPolicy`, and `Advanced`. • **securityValidationEnabled**—Boolean. Specifies whether security validation is enabled. • **stageMode**—The staging mode for the application you are deploying. Valid values are `stage`, `nostage`, and `stage_default`. For information about the staging modes, see "Staging Modes" in "Overview of WebLogic Server Deployment" in *Deploying WebLogic Server Applications*.This argument defaults to null. • **testMode**—Boolean. Specifies whether to start the Web application with restricted access.This argument defaults to `false`.

Example

The following example deploys the `myApp` application in the archive file located at `c:/myapps/myapp.ear` to `myserver` using the deployment plan file located in `c:/myapps/plan.xml`. Based on the `nostage` option set, the Administration Server does not copy deployment unit files to each target server. Instead, all servers deploy using the same physical copy of the deployment files, which must be directly accessible by the Administration Server and target servers.

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('myApp','c:/myapps/myapp.ear','myserver’, ‘nostage’,
c:/myapps/plan.xml’)
Deploying myApp from c:/myapps/myapp.ear ...
...Deployment of ‘myApp’ is successful
```

This 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.

distributeApplication

Command Category: Deployment Commands
Use with WLST: Online

Description

Copies the complete deployment bundle, module, configuration data, and any additional generated code to the specified targets (but 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.

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.
`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-17.

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.
```

This 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.

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
```

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.

Syntax

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

Argument	Definition
appInstallDir	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. Specifies 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>
```

This 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 `WLSTProgress` object, see "WLSTProgress Object" on page C-4.

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.

For more information about redeploying applications, see "Common Deployment Operations" in "Introduction to WebLogic Server Deployment" in *Deploying WebLogic Server Applications*.

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-17.

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>
```

This 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.

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-17.

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.
```

This 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.

Syntax

`stopApplication(`*`appName`*`, [`*`block`*`])`

Argument	Definition
appName	Name of the application to stop, as specified in the `plan.xml` file.
block	Optional. Boolean. Specifies 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.

Example

The following example stops the `BigApp` application.

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

This 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 or not `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.

For more information about undeploying applications, see "Common Deployment Operations" in *Deploying WebLogic Server Applications*.

Syntax

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

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.

Example

The following example removes the `myApp` application from all target servers.

```
wls:/mydomain/serverConfig> undeploy('myApp')
Undeployment of 'myApp' is successful
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.

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-17.

Example

The following example updates the application configuration for `BigApp` using the `plan.xml` file located in `c:/myapps/BigApp/newPlan`.

```
wls:/offline> progress=stopApplication('BigApp',
'c:/myapps/BigApp/newPlan/plan.xml', stageMode='STAGE', testMode='false')
Updating the application...
Successfully updated the application.
```

This 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 Command

Use the WLST diagnostics command, listed in Table A-5, to execute a query against a log file.

Table A-5 Diagnostic Command for WLST Configuration

This command...	Enables you to...	Use with WLST...
"exportDiagnosticData" on page A-28	Execute a query against the specified log file.	Online

exportDiagnosticData

Command Category: Diagnostics Command
Use with WLST: Online

Description

Executes a query against the specified log file. The results are saved to the XML file, `export.xml`.

Syntax

```
exportDiagnosticData(logType, logName, query, [beginTimeStamp],
[endTimeStamp])
```

```
exportDiagnosticData(logType, logName, logRotationDir, query,
[beginTimeStamp], [endTimeStamp])
```

Argument	Definition
logType	Type of the log file to be read.
logName	Name of the log file or store directory to be read.
query	Expression specifying the filter condition for the data records to be included in the result set.
beginTimestamp	Optional. Timestamp (inclusive) of the earliest record to be added to the result set.
endTimestamp	Optional. Timestamp (exclusive) of the latest record to be added to the result set.
logRotationDir	Directory containing the archived rotated log files.

Example

The following example exports all data from the server log.

```
wls:/mydomain/> exportDiagnosticData("ServerLog", "myserver.log", "")
```

Edit Commands

Use the WLST edit 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. 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 4-11.

Table A-6 Edit Commands for WLST Configuration

This command...	Enables you to...	Use with WLST...
"activate" on page A-31	Activate the changes made during the current editing session that have been saved to disk, but have not yet been deployed and made available for clients to use.	Online or Offline
"assign" on page A-32	Assign configuration beans to one or more destinations.	Offline
"assignAll" on page A-34	Assign all applications or services to one or more destinations.	Offline
"cancelEdit" on page A-35	Cancels an edit session and releases the edit lock.	Online
"create" on page A-35	Create a configuration bean for the current configuration bean.	Online or Offline
"delete" on page A-37	Delete an instance of a configuration for the current configuration bean.	Online or Offline
"get" on page A-37	Return the value of the specified attribute for the current configuration bean.	Online or Offline
"getActivationTask" on page A-38	Return the latest `ActivationTask` MBean on which a user can get status.	Online
"getMBean" on page A-39	Return the MBean by browsing to the specified path.	Online
"invoke" on page A-39	Invoke a management operation that an MBean exposes for its underlying resource.	Online
"isRestartRequired" on page A-40	Specifies whether or not a server restart is required, based on the changes that have been made during the current WLST session. The command prints the changes that require the server to be restarted.	Online
"loadDB" on page A-41	Load SQL files into a database.	Offline
"loadProperties" on page A-42	Get and set values from a properties file.	Online
"save" on page A-43	Save the edits that have been made but have not yet been saved.	Online

Table A-6 Edit Commands for WLST Configuration (Continued)

This command...	Enables you to...	Use with WLST...
"set" on page A-43	Set the specified attribute value for the current configuration bean.	Online or Offline
"setOption" on page A-44	Set options related to a domain creation or update	Offline
"showChanges" on page A-46	Show all of the changes that have been made by the current user to the configuration during the edit session.	Online
"startEdit" on page A-47	Start a configuration edit session.	Online
"stopEdit" on page A-47	Stop the current configuration edit session.	Online
"unassign" on page A-48	Unassign configuration beans from one or more destinations.	Offline
"unassignAll" on page A-50	Unassign all applications or services from one or more destinations.	Offline
"undo" on page A-51	Revert all edits since the last `save()` operation, discarding any edits that have not been saved.	Online
"validate" on page A-51	Validate the changes that have been made but have not yet been saved.	Online

activate

Command Category: Edit Commands
Use with WLST: Online

Description

Activates the changes made during the current editing session that have been saved to disk, but have not yet been deployed and made available for clients to use. 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.

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 0 indicates that the operation will not time out. This argument defaults to 60000 ms.
block	Optional. Boolean. Specifies 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.

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
successful.
wls:/mydomain/edit>
```

assign

Command Category: Edit Commands
Use with WLST: Offline

Description

Assigns configuration beans to one or more destinations.

Syntax

```
assign(sourceType, sourceName, destinationType, destinationName)
```

Argument	Definition
sourceType	Type of configuration bean to be assigned. This value can be set to `Application`, `Server`, or the name of a security type (such as `User`) or service (such as `JDBCConnectionPool`). Guidelines for setting this value are provided below.
sourceName	Name of the configuration bean to be assigned. Multiple names must be separated by commas.
destinationType	Type of destination. Guidelines for setting this value are provided below.
destinationName	Name of the destination. Multiple names must be separated by commas.

Use the following guidelines for setting the `sourceType` and `destinationType`:

- When assigning **applications** or **application components**, set the values as follows:
  - *sourceType*: `Application`
  - *destinationType*: `Target`
- When assigning **services**, set the values as follows:
  - *sourceType*: Name of the specific server, such as `JDBCConnectionPool`
  - *destinationType*: `Target`
- When assigning **servers** to **clusters**, set the values as follows:
  - *sourceType*: `Server`
  - *destinationType*: `Cluster`
- 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 example:

- Assigns the servers `myServer` and `myServer2` to the cluster `myCluster`.
- Assigns the application `MedRecEAR` to the target server `newServer`.

- Assigns the user `newUser` to the group `Monitors`.

```
wls:/offline/mydomain> assign("Server", "myServer,myServer2", "Cluster",
"myCluster")
wls:/offline/mydomain> assign("Application", "MedRecEAR", "Target",
"newServer")
wls:/offline/mydomain> assign("User", "newUser", "Group", "Monitors")
```

assignAll

Command Category: Edit Commands
Use with WLST: Offline

Description

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 "create" on page A-35.

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 must be 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: Edit Commands
Use with WLST: Online

Description

Cancels an edit session and releases the edit lock, enabling other users to start an edit session. Changes made since the last save are discarded. The user issuing this command does not have to be the current editor; this allows an administrator to cancel an edit session.

Syntax

`cancelEdit([`*`defaultResponse`*`])`

Argument	Definition
`defaultResponse`	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.

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
wls:/mydomain/edit>
```

create

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

Description

Creates a configuration bean for the current configuration bean. The `create` command returns a stub for the newly created configuration bean.

Note: 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`, see "WLST Variable Reference" on page A-95.

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 Configuration Edit MBean hierarchy using the `edit` command before issuing this command.
- 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:

- Use of periods ('.') within configuration bean names may produce unexpected results.

For more information about creating MBeans, see "Commands for Managing WebLogic Server MBeans" in *WebLogic Server Command Reference*.

Syntax

`create(`*`name, childBeanType`*`)`

Argument	Definition
`name`	Name of the configuration bean that you are creating.
`childBeanType`	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 Configuration Reference*.

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/serverConfig> server1=create('newServer','Server')
Server with name ‘myServer1’ has been created successfully.
wls:/mydomain/serverConfig> server1.getName()
‘newServer’
```

delete

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

Description

Deletes an instance of a configuration bean for the current configuration bean.

Note: You can only delete configuration beans that are children of current Configuration Management Object (`cmo`) type. For more information about the `cmo`, see “WLST Variable Reference” on page A-95.

Syntax

```
delete(name, childBeanType)
```

Argument	Definition
name	Name of the child configuration bean to delete.
childBeanType	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 Configuration Reference.*

Example

The following example deletes the configuration bean of type `Server` named `newServer`:

```
wls:/mydomain/serverConfig> delete('newServer','Server')
Server with name 'newServer' has been deleted successfully.
```

get

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

Description

Returns the value of the specified attribute for the current configuration bean.

Note: You can list all attributes and their current values by entering `ls('a')`. For more information, see "ls" on page A-59.

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 "WLST Variable Reference" on page A-95.

Syntax

get(*attributeName*)

Argument	Definition
attributeName	Name of the attribute to be displayed.

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: Edit Commands
Use with WLST: Online

Description

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.

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
```

getMBean

Command Category: Edit Commands
Use with WLST: Online

Description

Returns the MBean by browsing to the specified path.

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')
```

invoke

Command Category: Edit 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.

You cannot use the `invoke` command when WLST is connected to a Managed Server instance.

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. The class objects are loaded through the same class loader that is used for loading the configuration bean on which the action is invoked.

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: Edit Commands
Use with WLST: Online

Description

Specifies whether or not a server restart is required, based on the changes that have been made during the current WLST session. The command prints the changes that require the server to be restarted.

If you invoke this command while an edit session is in progress, the response is based on all edits that are currently in progress.

Syntax

`isRestartRequired([`*`attributeName`*`])`

Argument	Definition
`attributeName`	Name of attribute for which you want to check if a server restart is required.

Example

The following example specifies whether or not 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 which require server re-start.
MBean Changed : mydomain:Name=mydomain,Type=Domain
Attributes changed : AutoConfigurationSaveEnabled
```

The following example specifies whether or not 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: Edit Commands
Use with WLST: Offline

Description

Loads SQL files into a database.

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.

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

Description

Gets and sets values from a properties file.

Syntax

```
loadProperties(propertiesFile)
```

Argument	Definition
propertiesFile	Properties file pathname.

Example

This example gets and sets the properties file values.

```
wls:/mydomain/serverConfig> loadProperties('c:/temp/myLoad.properties')
```

save

Command Category: Edit Commands
Use with WLST: Online

Description

Saves the edits that have been made but have not yet been saved.

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: Edit Commands
Use with WLST: Online or Offline

Description

Sets the specified attribute value for the current configuration bean.

Note: You can list all attributes and their current values by entering `ls('a')`. For more information, see "ls" on page A-59.

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 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 "WLST Variable Reference" on page A-95.

Syntax

```
set(attributeName, attributeValue)
```

Argument	Definition
attributeName	Name of the attribute to be set.
attributeValue	Value of the attribute to be set. **Note:** This value does not need to 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)
wls:/mydomain/serverConfig>
```

setOption

Command Category: Edit Commands
Use with WLST: Offline

Description

Sets options related to a domain creation or update.

Syntax

```
setOption(optionName, optionValue)
```

Argument	Definition
optionName	Name of the option to set. Available options for **domain creation** include: • `CreateStartMenu`—specifies whether to create a Start Menu shortcut on a Windows platform. • `JavaHome`—specifies the home directory for the JVM to be used when starting the server. • `OverwriteDomain`—specifies whether to allow an existing domain to be overwritten. The default is `false`. • `ServerStartMode`—specifies the mode to use when starting the server for the newly created domain. This value can be `dev` (development) or `prod` (production). Available options for **domain updates** include: • `AllowCasualUpdate`—specifies whether allow a domain to be updated without adding an extension template. The default is `true`. • `ReplaceDuplicates`—specifies 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. Available options for both **domain creation** and **domain updates** include: • `AppDir`—specifies the application directory to be used when a separate directory is desired for applications, as specified by the template. • `AutoDeploy`—specifies 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.
optionValue	Value for the option.

Example

The following example sets the `CreateStartMenu` option to `false`:

```
wls:/offline> setOption('CreateStartMenu', 'false')
```

showChanges

Command Category: Edit Commands
Use with WLST: Online

Description

Shows all of the changes that have been made by the current user to the configuration during the edit session.

Syntax

`showChanges([onlyInMemory])`

Argument	Definition
onlyInMemory	Optional. Boolean. Specifies 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             : mydomain:Name=mydomain,Type=Domain
Operation Invoked         : add
Attribute Modified        : Servers
Attributes Old Value      : null
Attributes New Value      : wlstServer
Server Restart Required : false

MBean Changed             : mydomain:Name=wlstServer,Type=Server
Operation Invoked         : modify
Attribute Modified        : ListenPort
Attributes Old Value      : null
Attributes New Value      : 9001
Server Restart Required : false
```

startEdit

Command Category: Edit Commands
Use with WLST: Online

Description

Starts a configuration edit session. You must navigate to the Configuration Edit MBean hierarchy using the `edit` command before issuing this command. For more information, see "edit" on page A-91.

This command must be called prior to invoking any command to modify the domain configuration.

Syntax

```
startEdit([waitTimeInMillis], [timeoutInMillis])
```

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.
timeoutInMillis	Optional. Timeout (in milliseconds) to release the edit lock. This argument defaults to -1 ms, indicating that this edit session never expires.

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: Edit Commands
Use with WLST: Online

Description

Stops the current edit session, releasing the edit lock and allowing other users to start an edit session. Any changes made since the last save are discarded.

Syntax

```
stopEdit([defaultResponse])
```

Argument	Definition
defaultResponse	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.

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: Edit Commands
Use with WLST: Offline

Description

Unassign configuration beans from one or more destinations.

Syntax

```
unassign(sourceType, sourceName, destinationType, destinationName)
```

Argument	Definition
sourceType	Type of configuration bean to be unassigned. This value can be set to `Application`, `Server`, or the name of a security type (such as `User`) or service (such as `JDBCConnectionPool`). Guidelines for setting this value are provided below.
sourceName	Name of the configuration bean to be unassigned. Multiple names must be separated by commas.
destinationType	Type of destination. Guidelines for setting this value are provided below.
destinationName	Name of the destination. Multiple names must be separated by commas.

Use the following guidelines for setting the `sourceType` and `destinationType`:

- When unassiging **applications** or **application components**, set the values as follows:
  - *sourceType*: `Application`
  - *destinationType*: `Target`
- When unassiging **services**, set the values as follows:
  - *sourceType*: Name of the specific server, such as `JDBCConnectionPool`.
  - *destinationType*: `Target`
- When assigning **servers** from **clusters**, set the values as follows:
  - *sourceType*: `Server`.
  - *destinationType*: `Cluster`.
- 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`.

Example

The following example unassigns the following resources:

- Unassigns the servers `myServer` and `myServer2` from the cluster `myCluster`.
- Unassigns the application `MedRecEAR` from the target server `newServer`.

- Unassigns the user `newUser` from the group `Monitors`.

```
wls:/offline/medrec> unassign("Server", "myServer,myServer2", "Cluster",
"myCluster")
wls:/offline/medrec> unassign("Application", "MedRecEAR", "Target",
"newServer")
wls:/offline/medrec> unassign("User", "newUser", "Group", "Monitors")
```

unassignAll

Command Category: Edit Commands
Use with WLST: Offline

Description

Unassigns all applications or services from one or more destinations.

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 must be 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: Edit Commands
Use with WLST: Online

Description

Reverts all edits since the last `save()` operation, discarding any edits that have not been saved.

Syntax

`undo([`*`unactivateChanges`*`], [`*`defaultResponse`*`])`

Argument	Definition
`unactivateChanges`	Optional. Boolean. Specifies whether you want to undo all unactivated changes. This argument defaults to `false`.
`defaultResponse`	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.

Example

The following example reverts all changes since the last `save()` operation. WLST prompts for verification before canceling.

```
wls:/mydomain/edit !> undo()
Sure you would like to undo your changes? (y/n)
y
Discarded your in-memory changes successfully.
wls:/mydomain/edit>
```

validate

Command Category: Edit 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.

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...
"configToScript" on page A-53	Converts an existing server configuration file (`config.xml`) to an executable WLST script	Online
"dumpStack" on page A-55	Display the stack trace from the last exception that occurred.	Online or Offline
"dumpVariables" on page A-55	Display all variables used by WLST, including their name and value.	Online or Offline
"find" on page A-56	Find MBeans and attributes in the current or specified hierarchy.	Online
"getConfigManager" on page A-57	Return the latest `ConfigurationManagerBean` MBean which manages the change process.	Online
"getMBI" on page A-58	Returns the MBeanInfo for the specified MBeanType or the `cmo` variable.	Online
"listChildTypes" on page A-58	List all the children MBeans that can be created or deleted for the `cmo` type.	Online

Table A-7 Information Commands for WLST Configuration (Continued)

This command...	Enables you to...	Use with WLST...
"ls" on page A-59	List all child beans or attributes for the current configuration or runtime bean.	Online or Offline
"redirect" on page A-62	Redirects WLST output to the specified filename.	Online
"removeWatch" on page A-63	Removes a watch that was previously defined.	Online
"showWatch" on page A-63	Shows all watches that are currently defined.	Online
"startRecording" on page A-64	Record all user interactions with WLST; useful for capturing commands.	Online or Offline
"state" on page A-64	Return the state of a server or cluster.	Online
"stopRecording" on page A-65	Stop recording WLST commands.	Online or Offline
"stopRedirect" on page A-66	Stops redirection of WLST output to a file, if in progress.	Online or Offline
"storeUserConfig" on page A-66	Create a user configuration file and an associated key file.	Online
"threadDump" on page A-67	Display a thread dump for the specified server, or the current server if none is specified.	Online or Offline
"watch" on page A-68	Adds a watch, or "listener," to the specified MBean.	Online

configToScript

Command Category: Information Commands
Use with WLST: Online

Converts an existing server configuration file (`config.xml`) to an executable WLST script. You can use the resulting script to re-create the resources on other servers.

When you run the generated script, it starts a new server instance, and enters WLST script commands to create the server resources.

WLST creates the following files to contain the encrypted passwords and keys used to decrypt the passwords, respectively, where *domain_name* specifies the name of the current domain: c2sConfig*domain_name* and c2sSecret*domain_name*.

Syntax

```
configToScript([domainDir],[scriptPath],[overwrite],[propertiesFile],[depl
oymentScript])
```

Argument	Definition
domainDir	Optional. Path to the domain directory that contains the config.xml file that you want to convert. This argument defaults to the current directory in which WLST is running.
scriptPath	Optional. Path to which you want to write the converted WLST script. This argument defaults to the current directory in which WLST is running.
overwrite	Optional. Boolean. Specifies whether the script file should be overwritten. This argument defaults to true, indicating that the script file is overwritten.
propertiesFile	Optional. Path to which you want to write the properties files. This argument defaults to the value of scriptPath.
createDeleteFile	Optional. Boolean. Specifies 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 server resources configured in a config.xml file to a WLST script config.py:

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

```
Converting resource ... DomainLogFilter\OAMdomainLogFilter
Converting resource ... Server\adminServer
Converting resource ... SSL\adminServer
Converting resource ... XMLRegistry\registry_default
Converting resource ...
XMLEntitySpecRegistryEntry\myXMLEntitySpecRegistryEntry
Converting resource ... XMLRegistryEntry\myXMLRegistryEntry
Converting resource ... DomainLogFilter\wnewDomainLogFilter
```

```
ConfigToScript Successfully completed:, 7 resources are converted.
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.

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.

Syntax

```
dumpVariables()
```

Example

This example displays all the current variables and their values.

```
wls:/mydomain/serverConfig> dumpVariables()
cmo            [Caching Stub]Proxy for mydomain:Name=mydomain,Type=Domain
connected                     true
domainName                    mydomain
username                      weblogic
debug                         weblogic
isAdminServer                 true
serverName                    myserver
version        WebLogic Server 9.0 Mon Oct 27 22:57:18 PST 2005 305727
WebLogic XMLX Module 9.0  Mon Oct 27 22:57:18 PST 2005 305727
recording                     false
caseSyntax                    false
exitonerror                   true
wls:/mydomain/serverConfig>
```

find

Command Category: Information Commands
Use with WLST: Online

Description

Finds MBeans and attributes in the current or specified hierarchy. WLST returns the pathname to the MBean that stores the attribute and/or attribute type, and its value.

Syntax

find([*name*], [*type*], [*tree*])

Argument	Definition
name	Optional. Name of the attribute to find.
type	Optional. Type of the attribute to find.
tree	Optional. MBean hierarchy in which to search. This argument defaults to the current hierarchy.

Example

The following example searches for an attribute named `javaCompiler` in the current configuration hierarchy.

```
wls:/myserver/serverConfig> find('JavaCompiler')
searching ...
/Servers/BEA_Server                    JavaCompilerPreClassPath     null
/Servers/BEA_Server                    JavaCompiler                 java
/Servers/BEA_Server                    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')
searching ...
/JMSRuntime/myserver.jms
wls:/mydomain/serverRuntime>
```

getConfigManager

Command Category: Edit Commands
Use with WLST: Online

Description

Returns the latest `ConfigurationManagerBean` MBean which manages the change process. You can then invoke methods to manage configuration changes across a domain.

Syntax

```
getConfigurationManager()
```

Example

The following example returns the latest `ConfigurationManagerBean` MBean and stores it within the task variable.

```
wls:/mydomain/serverConfig> cm=getConfigurationManager()
```

getMBI

Command Category: Information Commands
Use with WLST: Online

Description

Returns the MBeanInfo for the specified MBeanType or the `cmo` variable.

Syntax

```
getMBI([mbeanType])
```

Argument	Definition
mbeanType	Optional. Specifies the MBeanType for which the MBeanInfo is displayed.

Example

The following example gets the MBeanInfo for the `cmo` and stores it in the variable `mbi`.

```
wls:/mydomain/serverConfig/> mbi=getMBI()
```

listChildTypes

Command Category: Information Commands
Use with WLST: Online

Description

Lists all the children MBeans that can be created or deleted for the `cmo`, the configuration bean instance to which you last navigated using WLST. For more information about `cmo`, see "WLST Variable Reference" on page A-95.

Syntax

```
listChildTypes([parentType])
```

Argument	Definition
parentType	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()
AppDeploymentsApplications
BridgeDestinations
CachingRealms
Capacities
...
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.

Table A-8 describes the `ls` command output information.

Table A-8 ls Command Output Information

Output	Definition
`d`	A 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.
`w`	Writeable property.
`x`	Executable operation.

Syntax

```
ls(['a' | 'c' | 'o'], [returnMap])
```

Argument	Definition
`a`	Optional. Display all the attribute names and values for the current configuration or runtime bean. If the attribute is encrypted, WLST displays six asterisks (`******`).
`c`	Optional. Display all the child configuration 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). Display all operations that can be invoked on the current Runtime MBean. Operations are not shown for Configuration MBeans.
`returnMap`	Optional. Boolean. If set to `true`, WLST returns 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. By default, WLST returns a String.

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/examples > ls()
drw-   Application
drw-   Cluster
drw-   EmbeddedLDAP
drw-   FileRealm
drw-   JDBCConnectionPool
drw-   JDBCTxDataSource
drw-   JMSConnectionFactory
drw-   JMSDistributedQueue
drw-   JMSDistributedTopic
drw-   JMSJDBCStore
drw-   JMSServer
drw-   JTA
drw-   MigratableTarget
drw-   PasswordPolicy
drw-   Realm
drw-   Security
drw-   SecurityConfiguration
```

```
drw-   Server
drw-   StartupClass

-rw-   AdministrationPort                                    9002
-rw-   AdministrationPortEnabled                             false
-rw-   AutoConfigurationSaveEnabled                          false
-rw-   ConfigurationVersion                                  9.0.0.0
-rw-   ConsoleContextPath                                    console
-rw-   ConsoleEnabled                                        true
-rw-   LastModificationTime                                  0
-rw-   Name                                                  examples
-rw-   Notes                                                 null
-rw-   ProductionModeEnabled                                 false
wls:/offline/examples>
```

The following example displays all the attribute names and values in `DomainMBean`:

```
wls:/mydomain/serverConfig> ls('a')
-rw-   AdministrationPort                                   9002
-rw-   AdministrationPortEnabled                            false
-rw-   ArchiveConfigurationCount                            10
-r--   CachingDisabled                                      true
-rw-   ClusterConstraintsEnabled                            false
-rw-   ConfigurationVersion                                 8.1.2.0
-rw-   ConsoleEnabled                                       true
-r--   DisconnectedManagedServers          [Ljava.lang.Object;@144b364
-rw-   EmbeddedLDAP         mydomain:Name=mydomain,Type=EmbeddedLDAP
-rw-  JTA                                mydomain:Name=mydomain,Type=JTA
-r--   LastModificationTime                                 0
-rw-   Log                               mydomain:Name=mydomain,Type=Log
-rw-   Name                                                 mydomain
-rw-   Notes                                                null
-rw-   ObjectName                  mydomain:Name=mydomain,Type=Domain
-rw-   Parent                                               null
-rw-   ProductionModeEnabled                                false
-r--   Registered                                           false
-r--   RootDirectory                                        .
-rw-   SNMPAgent                   mydomain:Name=mydomain,Type=SNMPAgent
-rw-   Security                    mydomain:Name=mydomain,Type=Security
```

```
-rw-   SecurityConfiguration
mydomain:Name=mydomain,Type=SecurityConfiguration
-rw-   Servers
mydomain:Name=managed2,Type=Server|mydomain:Name=managed8,Type=Server|mydo
main:Name=managed1,Type=Server|mydomain:Name=myserver,Type=Server|mydomain
:Name=managed7,Type=Server|mydomain:Name=managed4,Type=Server|mydomain:Nam
e=managed9,Type=Server|mydomain:Name=managed5,Type=Server|mydomain:Name=ma
naged6,Type=Server|mydomain:Name=managed3,Type=Server|mydomain:Name=manage
dServer,Type=Server|mydomain:Name=managed10,Type=Server
-r--   Type                                         Domain
-rw-   WTCServers
wls:/mydomain/serverConfig>
```

redirect

Command Category: Information Commands
Use with WLST: Online

Description

Redirects WLST output to the specified filename.

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. Specifies whether or not to the output should be sent to `stdout`. This argument defaults to `true`.

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')
```

removeWatch

Command Category: Information Commands
Use with WLST: Online

Description

Removes a watch that was previously defined. If you do not specify an argument, WLST removes all watches defined for all MBeans.

Syntax

```
removeWatch([mbean], [watchName])
```

Argument	Definition
mbean	Optional. Name of the MBean or MBean object for which you want to remove the previously defined watches.
recordAll	Optional. Name of the watch to be removed.

Example

The following example removes the watch named `mywatch`.

```
wls:/mydomain/serverConfig> removeWatch(watchName="mywatch")
```

showWatch

Command Category: Information Commands
Use with WLST: Online

Description

Shows all watches that are currently defined.

Syntax

```
showWatch()
```

Example

The following example shows all watches that are currently defined.

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

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.

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 enter the command.
recordAll	Optional. Boolean. Specifies whether or not 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
```

state

Command Category: Information Commands
Use with WLST: Online

Description

Returns the state of a server or cluster.

For more information about server states, see "Understanding Server Life Cycle" in *Managing Server Startup and Shutdown*.

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`.

Example

The following example returns the state of a Managed Server.

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

stopRecording

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

Description

Stops recording WLST commands.

Syntax

```
stopRecording()
```

Example

The following example stops recording WLST commands.

```
wls:/mydomain/serverConfig> stopRecording()
Stopping recording c:\myScripts\record.py
```

stopRedirect

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

Description

Stops the WLST output redirection to a file, if in progress.

Syntax

```
startRecording()
```

Example

The following example stops the redirection of WLST output to a file:

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

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.

Syntax

```
storeUserConfig(userConfigFile,userKeyFile)
```

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.

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:/temp/mysuserconfigfile.secure and c:/temp/myuserkeyfile.secure
wls:/mydomain/serverConfig>
```

threadDump

Command Category: Information Commands
Use with WLST: Online

Description

Displays a thread dump for the specified server, or the current server if none is specified.

Syntax

```
threadDump([writeToFile], [fileName], [serverName])
```

Argument	Definition
writeToFile	Optional. Boolean. Specifies whether or not you want 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')
```

watch

Command Category: Information Commands
Use with WLST: Online

Description

Adds a watch, or "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.

Syntax

```
watch(mbean, [attributeNames], [logFile], [watchName])
```

Argument	Definition
mbean	Name of the MBean or MBean object to watch.
attributeNames	Optional. Comma-separated list of all attribute names on which you would like to add a watch. This argument defaults to null, and adds a watch for all attributes.
logFile	Optional. Name and location of the log file to which you want to write watch information.This argument defaults to standard out.
watchName	Optional. Name of the watch. This argument defaults to a WLST-generated name.

Example

The following example defines a watch on the `cmo` MBean for the `Notes` and `ArchiveConfigurationCount` attributes. The watch is named `domain-watch` and is stored in `./watches/domain.log`.

```
wls:/mydomain/serverConfig> watch(cmo,
"Notes,ArchiveConfigurationCount","./watches/domain.log","domain-watch")
```

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...
"migrateAll" on page A-70	Migrate all JTA and JMS services to a target server in a cluster.	Online
"migrateServer" on page A-71	Migrate a server from one machine to another.	Online

Table A-9 Life Cycle Commands for WLST Configuration (Continued)

This command...	Enables you to...	Use with WLST...
"resume" on page A-72	Make a server available to receive requests from external clients.	Online
"shutdown" on page A-73	Gracefully shut down a running server instance or cluster.	Online
"start" on page A-74	Start a Managed Server instance or a cluster using a configured Node Manager.	Online
"startServer" on page A-75	Start the Administration Server.	Online or Offline
"suspend" on page A-76	Make a server unavailable to receive requests from external clients.	Online

migrateAll

Command Category: Life Cycle Commands
Use with WLST: Online

Description

Migrates all JTA and JMS services to a targeted server within a cluster.

Syntax

```
migrateServer(serverName, destinationName, [sourceDown],
[destinationDown])
```

Argument	Definition
serverName	Name of the server from which the JTA and JMS services should be migrated.
destinationName	Name of the machine to which you want to migrate the services.
sourceDown	Optional. Boolean. Specifies whether or not the source server is down. This argument defaults to `false`, indicating that the destination server is running.

Argument	Definition (Continued)
destinationDown	Optional. Boolean. Specifies whether or not the destination server is down. This argument defaults to `false`, indicating that the destination server is running.

Example

The following example migrates the 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 !> migrateServer('server1','server2', 'true', 'false')
Migrating server 'server1' to destination 'server2' ...
```

migrateServer

Command Category: Life Cycle Commands
Use with WLST: Online

Description

Migrates a server from one machine to another.

Syntax

```
migrateServer(serverName, machineName, [sourceDown], [destinationDown])
```

Argument	Definition
serverName	Name of the server to migrate.
machineName	Name of the machine to which you want to migrate the new server.
sourceDown	Optional. Boolean. Specifies whether or not the source server is down. This argument defaults to `false`, indicating that the destination server is running.
destinationDown	Optional. Boolean. Specifies whether or not the destination server is down. This argument defaults to `false`, indicating that the destination server is running.

Example

The following example migrates `myserver` to the machine `myUnixMachine`. The boolean arguments specify that the source server is down and the destination server is running.

```
wls:/mydomain/edit !> migrateServer('myserver','myUnixMachine', 'true',
'false')
Migrating server 'myserver' to machine 'myUnixMachine' ...
```

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*.

Syntax

```
resume([serverName], [block])
```

Argument	Definition
serverName	Name of the server to resume. This argument defaults to the server to which WLST is currently connected.
block	Optional. Boolean. Specifies 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.

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. This 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.

Syntax

```
shutdown([name],[type],[ignoreSessions],[timeout],[force])
```

Argument	Definition
name	Optional. Name of the server or cluster to shutdown. This argument defaults to the server to which WLST is currently connected.
type	Optional. Type, `Server` or `Cluster`. This argument defaults to `Server`.
ignoreSessions	Optional. Boolean. Specifies 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. Specifies 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.

Example

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)
```

The following example instructs WLST to drop all HTTP sessions immediately while connected to a Managed Server instance:

```
wls:/mydomain/serverConfig> shutdown('m1','Server','true',1200)
Shutting down a managed server that you are connected to ...
Disconnected from weblogic server: m1
wls:/(Not Connected)>
The scripting shell lost connection to the server that you were connected
to, this may be because the server was shutdown or partitioned. You will
have to re-connect to the server once the server is available.
```

start

Command Category: Life Cycle Commands
Use with WLST: Online or Offline

Description

Starts a Managed Server instance or a cluster using a configured and running Node Manager.

For more information about WLST commands used to connect to and use Node Manager, see "Node Manager Commands" on page A-77.

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`.
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`.

Argument	Definition (Continued)
block	Optional. Boolean. Specifies 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.

Example

The following example instructs the 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' ...
The server 'myserver' started successfully.
wls:/mydomain/serverConfig>
```

startServer

Command Category: Life Cycle Commands
Use with WLST: Online or Offline

Description

Starts the Administration Server.

Syntax

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

Argument	Definition
adminServerName	Optional. Name of the Administration Server to start. This argument defaults to `myserver`.
domainName	Optional. Name of the domain to which the Administration Server belongs. This argument defaults to `mydomain`.

Argument	Definition (Continued)
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. Specifies 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.
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`.

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')
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*.

Note: The Domain Administration port must be enabled to invoke the `suspend` command.

Syntax

```
suspend([serverName], [ignoreSessions], [timeout], [force], [block])
```

Argument	Definition
serverName	Optional. Name of the server to suspend. The argument defaults to the server to which WLST is currently connected.
ignoreSessions	Optional. Boolean. Specifies 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. Specifies 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. Specifies 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.

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 remote WebLogic Server instances. For more information about the Node Manager, see "Using Node Manager to Control Servers" in *Designing and Configuring WebLogic Server Environments*.

Note: Before you can use the Node Manager, you must start it, as described in "Starting and Stopping Node Manager" in *Designing and Configuring WebLogic Server Environments*.

Table A-10 Node Manager Commands for WLST Configuration

This command...	Enables you to...	Use with WLST...
"nm" on page A-78	Determine whether or not WLST is connected to a Node Manager.	Online
"nmConnect" on page A-79	Connect WLST to the Node Manager to establish a session.	Online or Offline
"nmDisconnect" on page A-80	Disconnect WLST from a Node Manager session.	Online
"nmEnroll" on page A-81	Enroll the machine on which WLST is currently running with the Node Manager.	Online
"nmKill" on page A-82	Kill the specified server instance that was started with the Node Manager.	Online
"nmLog" on page A-83	Return the Node Manager log.	Online
"nmServerLog" on page A-83	Return the server output log of the server that was started with the Node Manager.	Online
"nmServerStatus" on page A-84	Return the status of the server that was started with the Node Manager.	Online
"nmStart" on page A-85	Start a server in the current domain using the Node Manager.	Online
"nmVersion" on page A-85	Return the Node Manager server version.	Online

nm

Command Category: Node Manager Commands
Use with WLST: Online

Description

Specifies whether or not WLST is connected to a Node Manager. Returns `true` or `false` and prints a descriptive message.

Syntax

```
nm()
```

Example

The following example indicates that WLST is currently connected to a 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 a 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 the Node Manager to establish a session. After connecting to the Node Manager, you can invoke any Node Manager commands via WLST.

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 or not WLST is connected to a Node Manager, as described in "nm" on page A-78.

Syntax

```
nmConnect(username, password, [host], [port], [domainName], [nmType])
```

Argument	Definition
username	Username of the operator who is connecting WLST to the Node Manager.
password	Password of the operator who is connecting WLST to the Node Manager.
host	Optional. Host name of the Node Manager. This argument defaults to `localhost`.
port	Optional. Port number of the Node Manager. This argument defaults to a value that is based on the Node Manager server type.
domainName	Optional. Name of the domain that you want to manage. This argument defaults to `mydomain`.
nmType	Type of the Node Manager server. Valid values include: • `ssh` for script-based SSH implementation • `rsh` for RSH implementation • `ssl` for Java-based SSL implementation • `plain` for plain socket Java-based implementation This argument defaults to `ssl`.

Example

The following example connects WLST to the 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', '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.

Syntax

```
nmDisconnect()
```

Example

The following example disconnects WLST from a Node Manager session.

```
wls:/nsm/oamdomain> nmDisconnect()
Successfully disconnected from Node Manager
```

nmEnroll

Command Category: Node Manager Commands
Use with WLST: Online

Description

Enrolls the machine on which WLST is currently running with the Node Manager. This command downloads the Node Manager secret file (`nm_password.properties`) from the Administration Server to the specified directory, which contains the encrypted username and password that is used for server authentication. This command also updates the `nodemanager.domains` file under the *`BEA_HOME`*`/common/nodemanager` directory with this new domain.

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.

WLST must be connected to an Administration Server to run this command; WLST does not need to be connected to the Node Manager.

You must run this command once per domain per machine.

Syntax

```
nmEnroll([domainDir])
```

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.

Example

The following example enrolls the current machine with the Node Manager and saves the Node Manager secret file to `c:/nmdomain/nm_password.properties`.

```
wls:/mydomain/serverConfig> nmEnroll('c:/nmdomain')
Enrolling this machine with the domain directory at C:\nmdomain....
Successfully enrolled this machine with the domain directory at C:\nmdomain
wls:/mydomain/serverConfig>
```

nmKill

Command Category: Node Manager Commands
Use with WLST: Online

Description

Kills the specified server instance that was started with the Node Manager. If you attempt to kill a server instance that was not started using the Node Manager, the command displays an error. WLST must be connected to the Node Manager to run this command.

Syntax

```
nmKill([serverName])
```

Argument	Definition
domainDir	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 the Node Manager to run this command.

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 the Node Manager. WLST must be connected to the Node Manager to run this command.

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`.
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 the Node Manager. WLST must be connected to the Node Manager to run this command.

Syntax

`nmServerStatus([serverName])`

Argument	Definition
domainDir	Optional. Name of the server for which you want to display the status. This argument defaults to `myserver`.

Example

The following example kills the server named `oamserver`.

```
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 the Node Manager.

Syntax

`nmStart([serverName], [props], [writer])`

Argument	Definition
serverName	Optional. Name of the server to be started.
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 the Node Manager.

```
wls:/nm/mydomain> nmStart("managed1")
Starting server managed1 ...
Server managed1 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 the Node Manager to run this command.

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 1.0.0
wls:/nm/oamdomain>
```

Tree Commands

Use the WLST tree commands, listed in Table A-11, to navigate to the hierarchy of MBeans.

Table A-11 Tree Commands for WLST Configuration

Use this command...	To...	Use with WLST...
"config" on page A-87	Navigates 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 for WebLogic Server 9.0. You should update your script to use the `serverConfig` command as described in "serverConfig" on page A-93.	Online
"custom" on page A-89	Navigates to the root of Custom MBeans that are registered in the server.	Online
"domainConfig" on page A-90	Navigates to the last MBean to which you navigated in the configuration MBeans in the Domain runtime hierarchy or to the root of the hierarchy, `DomainMBean`.	Online
"domainRuntime" on page A-90	Navigates to the last MBean to which you navigated in the runtime MBeans in the Domain runtime hierarchy or to the root of the hierarchy, `DomainRuntimeMBean`.	Online

Table A-11 Tree Commands for WLST Configuration (Continued)

Use this command...	To...	Use with WLST...
"edit" on page A-91	Navigate to the last MBean to which you navigated in the Configuration Edit MBean hierarchy or to the root of the hierarchy, `DomainMBean`.	Online
"runtime" on page A-92	Navigates 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 for WebLogic Server 9.0. You should update your scripts to use the `serverRuntime` command, as described in "serverRuntime" on page A-94.	Online
"serverConfig" on page A-93	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-94	Navigates 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 for WebLogic Server 9.0. You should update your scripts to use the `serverConfig` command, as described in "serverConfig" on page A-93.

Navigates to the last MBean to which you navigated in the Administration or Local Configuration MBean hierarchy or to the root of the hierarchy, `DomainMBean`. The following table describes the result of running the `config` command when connected to an Administration or Managed Server.

When connected to...	The config command navigates to...

An Administration Server instance	Root of the Administration beans
A Managed Server instance	Root of the Local Configuration beans

For more information, see "Navigating Among MBean Hierarchies" on page 4-8.

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/serverConfig> ls()
drw-   Applications
drw-   BasicRealms
drw-   BridgeDestinations
drw-   CachingRealms
drw-   Clusters
drw-   CustomRealms
drw-   Deployments
drw-   DomainLogFilters
drw-   EmbeddedLDAP
drw-   FileRealms
drw-   FileT3s
drw-   JDBCConnectionPools
...
```

The following example navigates from the Runtime MBean hierarchy to the Configuration MBean hierarchy and back, on an Administration Server instance:

```
wls:/mydomain/runtime> cd('ServerRuntimes/myserver')
wls:/mydomain/runtime/ServerRuntimes/myserver> 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/serverConfig> cd('Servers/myserver/SSL/myserver')
wls:/mydomain/serverConfig/Servers/myserver/SSL/myserver> 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/ServerRuntimes/myserver>
```

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.

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 "Non-WebLogic Server MBeans" in *Programming WebLogic Management Services with JMX*.

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.

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()
wls:/mydomain/domainConfig> ls()
dr--   AppDeployments
dr--   Applications
dr--   BridgeDestinations
dr--   CachingRealms
dr--   Capacities
dr--   Clusters
dr--   Deployments
dr--   DomainLogFilters
dr--   EmbeddedLDAP
dr--   ErrorHandlings
dr--   FileRealms
dr--   FileStores
dr--   FileT3s
...
```

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.

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--   DeployerRuntime
dr--   ServerLifecycleRuntimes
dr--   ServerRuntimes
-r--  ActivationTime                              Tue Aug 03 08:58:38 EDT 2005
-r--   CachingDisabled                                  true
-r--   Clusters                                         null
-rw-   CurrentClusterDeploymentTarget                   null
...
```

edit

Command Category: Tree Commands
Use with WLST: Online

Description

Navigates to the last MBean to which you navigated in the Configuration Edit 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 4-11.

Syntax

```
edit()
```

Example

The following example illustrates how to navigate from the server Configuration MBean hierarchy to the editable copy of the domain Configuration Edit 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()
drw-   AppDeployments
drw-   Applications
drw-   BridgeDestinations
drw-   CachingRealms
drw-   Capacities
```

runtime

Command Category: Tree Commands
Use with WLST: Online

Description

Note: This command is deprecated for WebLogic Server 9.0. You should update your scripts to use the `serverRuntime` command, as described in "serverRuntime" on page A-94.

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`.

For more information, see "Browsing the Runtime Information" 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/ServerRuntime/managed1>
```

The following example navigates from the Configuration MBean hierarchy to the Runtime MBean hierarchy and back, on a Managed Server instance.

```
wls:/mydomain/serverConfig/ServersConfig/managed1> 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/ServerRuntime/managed1> cd('JVMRuntime/managed1')
wls:/mydomain/runtime/ServerRuntime/managed1/JVMRuntime/managed1'>
config()
wls:/mydomain/serverConfig/ServersConfig/managed1>
```

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 your current connected server. The MBean attribute values include any command-line overrides that a user specified while starting the server.

For more information, see "Navigating Among MBean Hierarchies" on page 4-8.

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 your current connected server.

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
`adminHome`	Represents Admin MBean home.
`autocommit`	Boolean. Specifies whether WLST automatically commits configuration changes as commands are entered. This variable defaults to `false`, indicating that all configuration changes that you make using WLST are treated as a set of modifications or a batch change; in this case, you enter the `save` command to commit the changes.
`cmo`	Stores the Current Management Object, which 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`.
`ct`	Reflects the state of the current change or change last completed.
`connected`	Boolean. Specifies 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`.
`domainName`	Stores the name of the domain to which WLST is connected.
`domainRuntimeService`	Represents the `weblogic.management.mbeanservers.domainruntime.DomainRuntimeServiceMBean` MBean.
`editService`	Represents the `weblogic.management.mbeanservers.edit.EditServiceMBean` MBean.
`exitonerror`	Boolean. Specifies whether WLST terminates script execution when WLST 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.

Table A-12 WLST Variables (Continued)

Variable	Description
`home`	Represents `MBeanHome`.
`isAdminServer`	Boolean. Specifies 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`.
`mbs`	Specifies the `MBeanServerConnection` object based on the current location in the hierarchy.
`recording`	Boolean. Specifies whether or not WLST is recording commands. WLST sets this variable to `true` when the `startRecording` command is entered; otherwise, WLST sets this variable to `false`.
`runtimeService`	Represents the `weblogic.management.mbeanservers.runtime.RuntimeServiceMBean` MBean.
`serverName`	Stores the name of the server to which WLST is connected.
`typeService`	Represents the `weblogic.management.mbeanservers.runtime.TypeServiceMBean` MBean.
`username`	Stores the username for WLST connection.
`version`	Stores the current version of the running server to which WLST is connected.

WLST Online and Offline Command Summary

The following sections summarize the WLST commands, as follows:

- "WLST Command Summary, Alphabetically By Command" on page B-1
- "WLST Online Command Summary" on page B-7
- "WLST Offline Command Summary" on page B-12

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-31	Activate the changes made during the current editing session that have been saved to disk, but have not yet been deployed and made available for clients to use.	Online
"addTemplate" on page A-8	Extend the current domain using the specified application or service extension template.	Offline
"assign" on page A-32	Assign configuration beans to one or more destinations.	Offline
"assignAll" on page A-34	Assign all applications or services to one or more destinations.	Offline

Table B-1 WLST Command Summary (Continued)

This command...	Enables you to...	Use with WLST...
"cancelEdit" on page A-35	Cancel an edit session and releases the edit lock.	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-9	Close the current domain template.	Offline
"config" on page A-87	Navigate to the last MBean to which you navigated in the Administration or Local Configuration MBean hierarchy or to the root of all configuration beans, `DomainMBean`. **Note:** This command is deprecated for WebLogic Server 9.0. You should update your script to use the `serverConfig` command as described in "serverConfig" on page A-93.	Online
"configToScript" on page A-53	Convert an existing server configuration file (`config.xml`) to an executable WLST script.	Online
"connect" on page A-10	Connect WLST to a WebLogic Server instance.	Online or Offline
"create" on page A-35	Create a configuration bean for the current configuration bean.	Online or Offline
"currentTree" on page A-4	Return the current location in the hierarchy.	Online
"custom" on page A-89	Navigate to the root of Custom MBeans that are registered in the server.	Online
"delete" on page A-37	Delete an instance of a configuration for the current configuration bean.	Online or Offline
"deploy" on page A-17	Deploy an application to a WebLogic Server instance.	Online
"disconnect" on page A-12	Disconnect WLST from a WebLogic Server instance.	Online
"distributeApplication" on page A-19	Copy the complete deployment bundle, module, configuration data, and any additional generated code to the specified targets.	Online

Table B-1 WLST Command Summary (Continued)

This command...	Enables you to...	Use with WLST...
"domainConfig" on page A-90	Navigates 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-90	Navigates 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-55	Display the 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-55	Display all variables used by WLST, including their name and value.	Online or Offline
"edit" on page A-91	Navigate to the last MBean to which you navigated in the Configuration Edit MBean hierarchy or to the root of the tree, `DomainMBean`.	Online
"exit" on page A-13	Exit WLST from the interactive session and close the scripting shell.	Online or Offline
"exportDiagnosticData" on page A-28	Execute a query against the specified log file.	Online
"find" on page A-56	Find MBeans and attributes in the current or specified hierarchy.	Online
"get" on page A-37	Return the value of the specified attribute for the current configuration bean.	Online or Offline
"getActivationTask" on page A-38	Return the latest `ActivationTask` MBean, which reflects the state of changes that a user is currently making or has made recently.	Online
"getConfigManager" on page A-57	Return the latest `ConfigurationManagerBean` MBean which manages the change process.	Online
"getMBean" on page A-39	Return the MBean by browsing to the specified path.	Online
"getMBI" on page A-58	Returns the MBeanInfo for the specified MBeanType or the `cmo` variable.	Online

Table B-1 WLST Command Summary (Continued)

This command...	Enables you to...	Use with WLST...
"getWLDM" on page A-21	Returns the WebLogic `DeploymentManager` object.	Online
"invoke" on page A-39	Invoke a management operation on the current configuration bean.	Online
"isRestartRequired" on page A-40	Specify whether or not a server restart is required, based on the changes that have been made during the current WLST session.	Online
"listChildTypes" on page A-58	Lists all the children MBeans that can be created or deleted for the `cmo`, the configuration bean instance to which you last navigated using WLST.	Online
"loadApplication" on page A-21	Load an application and deployment plan into memory.	Online or Offline
"loadDB" on page A-41	Load SQL files into a database.	Offline
"loadProperties" on page A-42	Get and set values from a properties file.	Online
"ls" on page A-59	List all child beans or attributes for the current configuration or runtime bean.	Online or Offline
"migrateAll" on page A-70	Migrate all JTA and JMS services to a target server in a cluster.	Online
"migrateServer" on page A-71	Migrate a server from one machine to another.	Online
"nm" on page A-78	Determine whether or not WLST is connected to a Node Manager.	Online
"nmConnect" on page A-79	Connect WLST to the Node Manager to establish a session.	Online
"nmDisconnect" on page A-80	Disconnect WLST from a Node Manager session.	Online
"nmEnroll" on page A-81	Enroll the machine on which WLST is currently running with the Node Manager.	Online
"nmKill" on page A-82	Kill the specified server instance that was started with the Node Manager.	Online

Table B-1 WLST Command Summary (Continued)

This command...	Enables you to...	Use with WLST...
"nmLog" on page A-83	Return the Node Manager log.	Online
"nmServerLog" on page A-83	Return the server output log of the server that was started with the Node Manager.	Online
"nmServerStatus" on page A-84	Return the status of the server that was started with the Node Manager.	Online
"nmStart" on page A-85	Starts a server in the current domain using the Node Manager.	Online
"nmVersion" on page A-85	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
"readDomain" on page A-13	Open an existing domain for updating.	Offline
"readTemplate" on page A-14	Open an existing domain template for domain creation.	Offline
"redeploy" on page A-22	Reload classes and redeploy a previously deployed application.	Online
"redirect" on page A-62	Redirects WLST output to the specified filename.	Online
"removeWatch" on page A-63	Removes a watch that was previously defined.	Online
"resume" on page A-72	Resumes a server instance that is suspended or in `ADMIN` state.	Online
"runtime" on page A-92	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 for WebLogic Server 9.0. You should update your scripts to use the `serverRuntime` command, as described in "serverRuntime" on page A-94.	Online

Table B-1 WLST Command Summary (Continued)

This command…	Enables you to…	Use with WLST…
"save" on page A-43	Save the edits that have been made but have not yet been saved.	Online
"serverConfig" on page A-93	Navigates 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-94	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-43	Set the specified attribute value for the current configuration bean.	Online or Offline
"setOption" on page A-44	Set options related to a domain creation or update	Offline
"showChanges" on page A-46	Show all of the changes that have been made by the current user to the configuration during the edit session.	Online
"showWatch" on page A-63	Shows all watches that are currently defined.	Online
"shutdown" on page A-73	Gracefully shut down a running server instance or cluster.	Online
"start" on page A-74	Start a Managed Server instance or a cluster using a configured and running Node Manager.	Online
"startApplication" on page A-24	Start an application, making it available to users.	Online
"startEdit" on page A-47	Start a configuration edit session.	Online
"startRecording" on page A-64	Record all user interactions with WLSTs.	Online or Offline
"startServer" on page A-75	Start the Administration Server.	Online or Offline
"state" on page A-64	Return the state of a server or cluster.	Online
"stopApplication" on page A-25	Stop an application, making it un available to users.	Online
"stopEdit" on page A-47	Stop the current edit session, releasing the edit lock and allowing other users to start an edit session.	Online

Table B-1 WLST Command Summary (Continued)

This command...	Enables you to...	Use with WLST...
"stopRecording" on page A-65	Stop recording WLST commands.	Online or Offline
"storeUserConfig" on page A-66	Create a user configuration file and an associated key file.	Online
"suspend" on page A-76	Suspend a running server.	Online
"threadDump" on page A-67	Display a thread dump for the specified server, or the current server if none is specified.	Online
"undeploy" on page A-26	Undeploy an application from the specified servers.	Online
"updateApplication" on page A-27	Update an application configuration using a new deployment plan.	Online
"updateDomain" on page A-14	Update and save the current domain.	Offline
"unassign" on page A-48	Unassign configuration beans from one or more destinations.	Offline
"unassignAll" on page A-50	Unassign all applications or services from one or more destinations.	Offline
"undo" on page A-51	Revert all edits since the last `save()` operation, discarding any edits that have not been saved.	Online
"validate" on page A-51	Validate the changes that have been made but have not yet been saved.	Online
"watch" on page A-68	Adds a watch, or "listener," to the specified MBean.	Online
"writeDomain" on page A-15	Write the domain configuration information to the specified directory.	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-31	Activate the changes made during the current editing session that have been saved to disk, but have not yet been deployed and made available for clients to use.
"cancelEdit" on page A-35	Cancel an edit session and releases the edit lock.
"cd" on page A-3	Navigate the hierarchy of configuration or runtime beans.
"config" on page A-87	Navigate to the last MBean to which you navigated in the Administration or Local Configuration MBean hierarchy or to the root of all configuration beans, `DomainMBean`. **Note:** This command is deprecated for WebLogic Server 9.0. You should update your script to use the `serverConfig` command as described in "serverConfig" on page A-93.
"configToScript" on page A-53	Convert an existing server configuration file (`config.xml`) to an executable WLST script.
"connect" on page A-10	Connect WLST to a WebLogic Server instance.
"create" on page A-35	Create a configuration bean for the current configuration bean.
"currentTree" on page A-4	Return the current tree location.
"custom" on page A-89	Navigate to the root of Custom MBeans that are registered in the server.
"delete" on page A-37	Delete an instance of a configuration for the current configuration bean.
"deploy" on page A-17	Deploy an application to a WebLogic Server instance.
"disconnect" on page A-12	Disconnect WLST from a WebLogic Server instance.
"distributeApplication" on page A-19	Copy the complete deployment bundle, module, configuration data, and any additional generated code to the specified targets.
"domainConfig" on page A-90	Navigates to the last MBean to which you navigated in the Domain Configuration hierarchy or to the root of the hierarchy, `DomainMBean`.
"domainRuntime" on page A-90	Navigates to the last MBean to which you navigated in the Domain Runtime hierarchy or to the root of the hierarchy, `DomainRuntimeMBean`.

Table B-2 WLST Online Command Summary (Continued)

This command...	Enables you to...
"dumpStack" on page A-55	Display the stack trace from the last exception that occurred while performing a WLST action, and reset the stack trace.
"dumpVariables" on page A-55	Display all variables used by WLST, including their name and value.
"edit" on page A-91	Navigate to the last MBean to which you navigated in the Configuration Edit MBean hierarchy or to the root of the tree, `DomainMBean`.
"exit" on page A-13	Exit WLST from the interactive session and close the scripting shell.
"exportDiagnosticData" on page A-28	Execute a query against the specified log file.
"find" on page A-56	Find MBeans and attributes in the current or specified hierarchy.
"get" on page A-37	Return the value of the specified attribute for the current configuration bean.
"getActivationTask" on page A-38	Return the latest `ActivationTask` MBean, which reflects the state of changes that a user is currently making or has made recently.
"getConfigManager" on page A-57	Return the latest `ConfigurationManagerBean` MBean which manages the change process.
"getMBean" on page A-39	Return the MBean by browsing to the specified path.
"getMBI" on page A-58	Return the MBeanInfo for the specified MBeanType or the `cmo` variable.
"getWLDM" on page A-21	Return the WebLogic `DeploymentManager` object.
"invoke" on page A-39	Invoke a management operation on the current configuration bean.
"isRestartRequired" on page A-40	Specify whether or not a server restart is required, based on the changes that have been made during the current WLST session.
"listChildTypes" on page A-58	Lists all the children MBeans that can be created or deleted for the `cmo`, the configuration bean instance to which you last navigated using WLST.
"loadApplication" on page A-21	Load an application and deployment plan into memory.
"loadProperties" on page A-42	Get and set values from a properties file.
"ls" on page A-59	List all child beans or attributes for the current configuration or runtime bean.

Table B-2 WLST Online Command Summary (Continued)

This command...	Enables you to...
"migrateAll" on page A-70	Migrate all JTA and JMS services to a target server in a cluster.
"migrateServer" on page A-71	Migrate a server from one machine to another.
"nm" on page A-78	Determine whether or not WLST is connected to a Node Manager.
"nmConnect" on page A-79	Connect WLST to the Node Manager to establish a session.
"nmDisconnect" on page A-80	Disconnect WLST from a Node Manager session.
"nmEnroll" on page A-81	Enroll the machine on which WLST is currently running with the Node Manager.
"nmKill" on page A-82	Kill the specified server instance that was started with the Node Manager.
"nmLog" on page A-83	Return the Node Manager log.
"nmServerLog" on page A-83	Return the server output log of the server that was started with the Node Manager.
"nmServerStatus" on page A-84	Return the status of the server that was started with the Node Manager.
"nmStart" on page A-85	Starts a server in the current domain using the Node Manager.
"nmVersion" on page A-85	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.
"redeploy" on page A-22	Reload classes and redeploy a previously deployed application.
"redirect" on page A-62	Redirects WLST output to the specified filename.
"removeWatch" on page A-63	Removes a watch that was previously defined.
"resume" on page A-72	Resumes a server instance that is suspended or in `ADMIN` state.

Table B-2 WLST Online Command Summary (Continued)

This command...	Enables you to...
"runtime" on page A-92	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 for WebLogic Server 9.0. You should update your scripts to use the `serverRuntime` command, as described in "serverRuntime" on page A-94.
"save" on page A-43	Save the edits that have been made but have not yet been saved.
"serverConfig" on page A-93	Navigates 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-94	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-43	Set the specified attribute value for the current configuration bean.
"showChanges" on page A-46	Show all of the changes that have been made by the current user to the configuration during the edit session.
"showWatch" on page A-63	Shows all watches that are currently defined.
"shutdown" on page A-73	Gracefully shut down a running server instance or cluster.
"start" on page A-74	Start a Managed Server instance or a cluster using a configured and running Node Manager.
"startApplication" on page A-24	Start an application, making it available to users.
"startEdit" on page A-47	Start a configuration edit session.
"startRecording" on page A-64	Record all user interactions with WLST.
"startServer" on page A-75	Start the Administration Server.
"state" on page A-64	Return the state of a server or cluster.
"stopApplication" on page A-25	Stop an application, making it un available to users.
"stopEdit" on page A-47	Stop the current edit session, releasing the edit lock and allowing other users to start an edit session.
"stopRecording" on page A-65	Stop recording WLST commands.

Table B-2 WLST Online Command Summary (Continued)

This command...	Enables you to...
"storeUserConfig" on page A-66	Create a user configuration file and an associated key file.
"suspend" on page A-76	Suspend a running server.
"threadDump" on page A-67	Display a thread dump for the specified server, or the current server if none is specified.
"undeploy" on page A-26	Undeploy an application from the specified servers.
"undo" on page A-51	Revert all edits since the last `save()` operation, discarding any edits that have not been saved.
"updateApplication" on page A-27	Update an application configuration using a new deployment plan.
"validate" on page A-51	Validate the changes that have been made but have not yet been saved.

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 the specified application or service extension template.
"assign" on page A-32	Assign configuration beans to one or more destinations.
"assignAll" on page A-34	Assign all applications or services to one or more destinations.
"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-9	Close the current domain template.
"connect" on page A-10	Connect WLST to a WebLogic Server instance.
"create" on page A-35	Create a configuration bean for the current configuration bean.
"delete" on page A-37	Delete an instance of a configuration for the current configuration bean.

Table B-3 WLST Offline Command Summary (Continued)

This command...	Enables you to...
"dumpStack" on page A-55	Display the stack trace from the last exception that occurred while performing a WLST action, and reset the stack trace.
"dumpVariables" on page A-55	Display all variables used by WLST, including their name and value.
"exit" on page A-13	Exit WLST from the interactive session and close the scripting shell.
"get" on page A-37	Return the value of the specified attribute for the current configuration bean.
"loadDB" on page A-41	Load SQL files into a database.
"ls" on page A-59	List all child beans or attributes for the current configuration or runtime bean.
"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-13	Open an existing domain for updating.
"readTemplate" on page A-14	Open an existing domain template for domain creation.
"set" on page A-43	Set the specified attribute value for the current configuration bean.
"setOption" on page A-44	Set options related to a domain creation or update.
"start" on page A-74	Start a Managed Server instance or a cluster using a configured and running Node Manager.
"startRecording" on page A-64	Record all user interactions with WLST.
"startServer" on page A-75	Start the Administration Server.
"stopRecording" on page A-65	Stop recording WLST commands.
"updateDomain" on page A-14	Update and save the current domain.
"unassign" on page A-48	Unassign configuration beans from one or more destinations.
"unassignAll" on page A-50	Unassign all applications or services from one or more destinations.
"writeDomain" on page A-15	Write the domain configuration information to the specified directory.

WLST Deployment Objects

The following sections describe the WLST deployment objects:

- "WLSTPlan Object" on page C-1
- "WLSTProgress Object" on page C-4

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-21.

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` *`module1OverrideName`*.
	`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-17
- "distributeApplication" on page A-19
- "redeploy" on page A-22
- "startApplication" on page A-24
- "stopApplication" on page A-25
- "updateApplication" on page A-27

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`.
`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.