This document describes the 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 WebLogic domains, and monitor and manage server run-time events.
Contents

Preface ................................................................................................................................................................... vii
  Documentation Accessibility ........................................................................................................ vii
  Conventions ................................................................................................................................................ vii

1 Introduction and Roadmap
  1.1 Document Scope and Audience ........................................................................................................ 1-1
  1.2 Guide to This Document ................................................................................................................... 1-1
  1.3 Related Documentation .................................................................................................................... 1-2
  1.4 WLST Sample Scripts ....................................................................................................................... 1-2
    1.4.1 WLST Online Sample Scripts .................................................................................................. 1-3
    1.4.2 WLST Offline Sample Scripts .................................................................................................. 1-3
  1.5 New and Changed WLST Features in This Release ....................................................................... 1-5

2 Using the WebLogic Scripting Tool
  2.1 Using WLST Online or Offline ......................................................................................................... 2-1
    2.1.1 Using WLST Online .................................................................................................................. 2-1
    2.1.2 Using WLST Offline ................................................................................................................ 2-2
  2.2 Interactive Mode, Script Mode, and Embedded Mode .................................................................... 2-2
    2.2.1 Interactive Mode ..................................................................................................................... 2-2
    2.2.2 Script Mode ............................................................................................................................. 2-3
    2.2.3 Embedded Mode ...................................................................................................................... 2-3
  2.3 Security for WLST ............................................................................................................................... 2-4
    2.3.1 Securing the WLST Connection ......................................................................................... 2-4
    2.3.2 Securing Access to Configuration Data ................................................................................ 2-5
      2.3.2.1 Securing Access from WLST Online ............................................................................. 2-5
      2.3.2.2 Writing and Reading Encrypted Configuration Values ........................................... 2-6
    2.3.3 Securing Access to Security Data ......................................................................................... 2-7
  2.4 Main Steps for Using WLST in Interactive or Script Mode ............................................................ 2-8
    2.4.1 Invoking WLST ......................................................................................................................... 2-8
      2.4.1.1 Invoking WLST Using Provided Shell Scripts ........................................................... 2-8
      2.4.1.2 Invoking WLST Using the java Command ............................................................. 2-9
      2.4.1.3 Running Scripts ............................................................................................................. 2-11
      2.4.1.4 Invoking WLST From the Start Menu ...................................................................... 2-12
    2.4.2 Exiting WLST ............................................................................................................................ 2-12
    2.4.3 Syntax for WLST Commands ................................................................................................. 2-12
Preface

This preface describes the document accessibility features and conventions used in this guide—WebLogic Server Scripting Tool.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at http://www.oracle.com/accessibility/.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/support/contact.html or visit http://www.oracle.com/accessibility/support.html if you are hearing impaired.

Conventions

The following text conventions are used in this document:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>boldface</td>
<td>Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.</td>
</tr>
<tr>
<td>Convention</td>
<td>Meaning</td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
</tr>
<tr>
<td>italic</td>
<td>Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.</td>
</tr>
<tr>
<td>monospace</td>
<td>Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.</td>
</tr>
</tbody>
</table>
1

Introduction and Roadmap

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

- Section 1.1, "Document Scope and Audience"
- Section 1.2, "Guide to This Document"
- Section 1.3, "Related Documentation"
- Section 1.4, "WLST Sample Scripts"
- Section 1.5, "New and Changed WLST Features in This Release"

1.1 Document Scope and Audience

This document describes the Oracle 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 WebLogic domains, and monitor and
manage server run-time events.

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

1.2 Guide to This Document

This document is organized as follows:

- This chapter, Chapter 1, "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 WebLogic Domains Using WLST Offline," describes how to
  create a new WebLogic domain or update an existing WebLogic domain without
  connecting to a running WebLogic Server (that is, using WLST
  offline)—supporting the same functionality as the Configuration Wizard.
- Chapter 4, "Managing the Server Life Cycle," describes using WLST to start and
  stop WebLogic Server instances and to monitor and manage the server life cycle.
- Chapter 5, "Navigating MBeans (WLST Online)," describes how to retrieve
  WebLogic domain configuration and run-time information, and edit configuration
  or custom MBeans.
Chapter 6, "Configuring Existing WebLogic Domains" describes using scripts to automate the creation and management of WebLogic domains, servers, and resources.

Chapter 7, "Updating the Deployment Plan," describes using WLST to update an application's deployment plan.

Chapter 8, "Getting Runtime Information," describes using WLST to retrieve information about the run-time state of WebLogic Server instances.

Appendix A, "WLST Deployment Objects," describes WLST deployment objects that you can use to update a deployment plan or access information about the current deployment activity.

Appendix B, "FAQs: WLST," provides a list of common questions and answers.

1.3 Related Documentation

Detailed information about Oracle WebLogic Server WLST commands, as well as custom commands for Oracle Fusion Middleware components, can be found in the WebLogic Scripting Tool Command Reference.

WLST is one of several interfaces for managing and monitoring WebLogic Server. For information about the other management interfaces, see:

- "Using Ant Tasks to Configure and Use a WebLogic Server Domain" in Developing Applications for Oracle WebLogic Server, describes using WebLogic Ant tasks for starting and stopping WebLogic Server instances and configuring WebLogic domains.
- "Deployment Tools" in Deploying Applications to Oracle WebLogic Server describes several tools that WebLogic Server provides for deploying applications and stand-alone modules.
- Oracle WebLogic Server Administration Console Help describes a Web-based graphical user interface for managing and monitoring WebLogic domains.
- Creating Domains Using the Configuration Wizard describes using a graphical user interface to create a WebLogic domain or extend an existing one.
- Creating Templates and Domains Using the Pack and Unpack Commands describes commands that recreate existing WebLogic domains quickly and easily.
- Developing Custom Management Utilities With JMX for Oracle WebLogic Server describes using Java Management Extensions (JMX) APIs to monitor and modify WebLogic Server resources.

1.4 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 Section 2.4.1.3, "Running Scripts".

Note: The sample scripts are not installed by default. To install the server samples, you must select a custom installation of WebLogic Server and select to install the Server Examples.
1.4.1 WLST Online Sample Scripts

The WLST online sample scripts demonstrate how to perform administrative tasks and initiate WebLogic Server configuration changes while connected to a running server. WLST online scripts are located in the following directory: `SAMPLES_HOME\server\examples\src\examples\wlst\online`, where `SAMPLES_HOME` refers to the main examples directory of your WebLogic Server installation, such as `c:\Oracle\Middleware\wlserver_10.3\samples`.

Table 1–1 summarizes WLST online sample scripts.

<table>
<thead>
<tr>
<th>WLST Sample Script</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cluster_creation.py</td>
<td>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.</td>
</tr>
<tr>
<td>cluster_deletion.py</td>
<td>Removes the clusters and servers created in <code>cluster_creation.py</code>.</td>
</tr>
<tr>
<td>configJMSSystemResource.py</td>
<td>Connects WLST to an Administration Server, starts an edit session, creates two JMS Servers, and targets them to the Administration Server. Then creates JMS topics, JMS queues, and JMS templates in a JMS System module. The JMS queues and topics are targeted using sub-deployments.</td>
</tr>
<tr>
<td>deleteJMSSystemResource.py</td>
<td>Removes the JMS System module created by <code>configJMSSystemResource.py</code>.</td>
</tr>
<tr>
<td>jdbc_data_source_creation.py</td>
<td>Connects WLST to an Administration Server, starts an edit session, and creates a JDBC data source called <code>myJDBCDataSource</code>.</td>
</tr>
<tr>
<td>jdbc_data_source_deletion.py</td>
<td>Removes the JDBC data source created by <code>jdbc_data_source_creation.py</code>.</td>
</tr>
</tbody>
</table>

1.4.2 WLST Offline Sample Scripts

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

Table 1–2 summarizes WLST offline sample scripts.

<table>
<thead>
<tr>
<th>WLST Sample Script</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>basicWLSdomain.py</td>
<td>Creates a simple WebLogic domain demonstrating how to open a domain template, create and edit configuration objects, and write the domain configuration information to the specified directory. The sample consists of a single server, representing a typical development environment. This type of configuration is not recommended for production environments. The script uses the Basic WebLogic Server Domain template.</td>
</tr>
</tbody>
</table>
Table 1–2 (Cont.) WLST Offline Sample Script

<table>
<thead>
<tr>
<th>WLST Sample Script</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>basicWLSSdomain.py</td>
<td>Creates a simple WebLogic SIP Server domain using the Basic WebLogic SIP 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. The sample consists of a single server, representing a typical development environment. This type of configuration is not recommended for production environments.</td>
</tr>
<tr>
<td>clusterMedRecDomain.py</td>
<td>Creates a single-cluster WebLogic domain, creating three Managed Servers and assigning them to a cluster. The script uses the Basic WebLogic Server Domain template and extends it using the Avitek Medical Records Sample extension template.</td>
</tr>
<tr>
<td>distributedQueue.py</td>
<td>Demonstrates two methods for creating distributed queues. The script uses the Basic WebLogic Server Domain template and extends it using the Avitek Medical Records Sample extension template.</td>
</tr>
<tr>
<td>geo1Domain.py</td>
<td>Creates a simple WebLogic SIP Server domain using the Geographic Redundancy Site 1 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. The sample consists of a single server, representing a typical development environment. This type of configuration is not recommended for production environments.</td>
</tr>
<tr>
<td>geo2Domain.py</td>
<td>Creates a simple WebLogic SIP Server domain using the Geographic Redundancy Site 2 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. The sample consists of a single server, representing a typical development environment. This type of configuration is not recommended for production environments.</td>
</tr>
<tr>
<td>replicatedDomain.py</td>
<td>Creates a simple WebLogic SIP Server domain using the Oracle WebLogic SIP Server Replicated 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. The sample consists of a single server, representing a typical development environment. This type of configuration is not recommended for production environments.</td>
</tr>
<tr>
<td>sampleMedRecDomain.py</td>
<td>Creates a WebLogic domain that defines resources similar to those used in the Avitek MedRec sample. This example does not recreate the MedRec example in its entirety, nor does it deploy any sample applications. The script uses the Basic WebLogic Server Domain template.</td>
</tr>
</tbody>
</table>

In addition, sample scripts are provided that configure WebLogic domain resources using WLST offline and online on the Oracle Technology Network site at [http://www.oracle.com/technology/index.html](http://www.oracle.com/technology/index.html). To locate WLST sample scripts and information about WLST scripts, search for any of the following terms:

- WLST scripts
- WLST online scripts
- WLST offline scripts

1.5 New and Changed WLST Features in This Release

For a comprehensive listing of the new WebLogic Server features introduced in this release, see What’s New in Oracle WebLogic Server.
Using the WebLogic Scripting Tool

The WebLogic Scripting Tool (WLST) is a command-line scripting environment that you can use to create, manage, and monitor WebLogic domains. It is based on the Java scripting interpreter, Jython. In addition to supporting standard Jython features such as local variables, conditional variables, and flow control statements, WLST provides a set of scripting functions (commands) that are specific to WebLogic Server. You can extend the WebLogic scripting language to suit your needs by following the Jython language syntax. See http://www.jython.org.

The following sections describe the WebLogic Scripting Tool:

- Section 2.1, "Using WLST Online or Offline"
- Section 2.2, "Interactive Mode, Script Mode, and Embedded Mode"
- Section 2.3, "Security for WLST"
- Section 2.4, "Main Steps for Using WLST in Interactive or Script Mode"
- Section 2.5, "Redirecting Error and Debug Output to a File"
- Section 2.6, "Getting Help"
- Section 2.7, "Running WLST from Ant"
- Section 2.8, "Importing WLST as a Jython Module"
- Section 2.9, "Customizing WLST"

2.1 Using WLST Online or Offline

You can use WLST as the command-line equivalent to the WebLogic Server Administration Console (WLST online) or as the command-line equivalent to the Configuration Wizard (WLST offline). For information about the WebLogic Server Administration Console, see Oracle WebLogic Server Administration Console Help. For information about the Configuration Wizard, see Creating Domains Using the Configuration Wizard.

2.1.1 Using WLST Online

You can use WLST to connect to a running Administration Server and manage the configuration of an active WebLogic domain, view performance data about resources in the domain, or manage security data (such as adding or removing users). You can also use WLST to connect to Managed Servers, but you cannot modify configuration data from Managed Servers.

WLST online is a Java Management Extensions (JMX) client. It interacts with a server's in-memory collection of Managed Beans (MBeans), which are Java objects that provide
Interactive Mode, Script Mode, and Embedded Mode

2.1.2 Using WLST Offline

Without connecting to a running WebLogic Server instance, you can use WLST to create domain templates, create a new domain based on existing templates, or extend an existing, inactive domain. You cannot use WLST offline to view performance data about resources in a WebLogic domain or modify security data (such as adding or removing users).

WLST offline provides read and write access to the configuration data that is persisted in the domain's `config` directory or in a domain template JAR created using Template Builder. See Section 3.1.1, "Browsing Information About the Configuration Hierarchy (Offline)" for more information.

Note the following restrictions for modifying configuration data with WLST offline:

- Oracle recommends that you do not use WLST offline to manage the configuration of an active WebLogic domain. Offline edits are ignored by running servers and can be overwritten by JMX clients such as WLST online or the WebLogic Server Administration Console.
- As a performance optimization, WebLogic Server does not store most of its default values in the WebLogic domain's configuration files. In some cases, this optimization prevents management objects from being displayed by WLST offline (because WebLogic Server has never written the corresponding XML elements to the domain's configuration files). For example, if you never modify the default logging severity level for a WebLogic domain while the domain is active, WLST offline will not display the domain's Log management object.

If you want to change the default value of attributes whose management object is not displayed by WLST offline, you must first use the `create` command to create the management object. Then you can `cd` to the management object and change the attribute value. See "create" in WebLogic Scripting Tool Command Reference.

2.2 Interactive Mode, Script Mode, and Embedded Mode

You can use any of the following techniques to invoke WLST commands:

- Interactively, on the command line; see Section 2.2.1, "Interactive Mode"
- In batches, supplied in a file; see Section 2.2.2, "Script Mode"
- Embedded in Java code; see Section 2.2.3, "Embedded Mode"

2.2.1 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.

WLST can write all of the commands that you enter during a WLST session to a file. You can edit this file and run it as a WLST script. For more information, see "startRecording" and "stopRecording" in WebLogic Scripting Tool Command Reference.
2.2.2 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.

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

For information about sample scripts that WebLogic Server installs, see Section 1.4, "WLST Sample Scripts".

2.2.3 Embedded Mode

In embedded mode, you instantiate 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.

Example 2–1 illustrates how to instantiate the WLST interpreter and use it to connect to a running server, create two servers, and assign them to clusters.

**Example 2–1 Running WLST From a Java Class**

```java
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: Oracle</p>
 */

public class EmbeddedWLST {
    static InteractiveInterpreter interpreter = null;
    EmbeddedWLST() {
        interpreter = new WLSTInterpreter();
    }

    private static void connect() {
        StringBuffer buffer = new StringBuffer();
        buffer.append("connect('weblogic','welcome1')");
        interpreter.exec(buffer.toString());
    }
}
```
2.3 Security for WLST

WLST uses the WebLogic Security Framework to prevent unauthorized users from modifying a WebLogic domain or from viewing encrypted data. The following sections describe the actions you must take to satisfy WLST security requirements:

- Section 2.3.1, "Securing the WLST Connection"
- Section 2.3.2, "Securing Access to Configuration Data"
- Section 2.3.3, "Securing Access to Security Data"

2.3.1 Securing the WLST Connection

If you use WLST to connect to a WebLogic Server instance, Oracle recommends that you connect to the server instance through the administration port. The administration port is a special, secure port that all WebLogic Server instances in a WebLogic domain can use for administration traffic.

By default, this port is not enabled, but Oracle recommends that you enable the administration port in a production environment. The default value for the administration port is 9002. Separating administration traffic from application traffic ensures that critical administration operations (starting and stopping servers,
changing a server’s configuration, and deploying applications) do not compete with high-volume application traffic on the same network connection.

The administration port requires all communication to be secured using SSL. By default, all servers in a WebLogic domain use demonstration certificate files for SSL, but these certificates are not appropriate for a production environment.

For information about configuring the administration port, see “Administration Port and Administrative Channel” in Configuring Server Environments for Oracle WebLogic Server.

2.3.2 Securing Access to Configuration Data

A WebLogic domain stores its configuration data in a collection of XML documents that are saved in the domain directory. For example, these configuration documents describe the names, listen addresses, and deployed resources in the domain. When one or more servers in a WebLogic domain are running, each server instance maintains an in-memory representation of the configuration data as a collection of Managed Beans (MBeans).

You must use your own security measures to make sure that only authorized users can access your domain’s configuration files through the file system. Anyone who is authorized to access the domain’s configuration files through the file system can use a text editor, WLST offline, or other tools to edit the configuration files.

2.3.2.1 Securing Access from WLST Online

If you use WLST to connect to a running instance of WebLogic Server, you must provide the credentials (user name and password) of a user who has been defined in the active WebLogic security realm. Once you are connected, a collection of security policies determine which configuration attributes you are permitted to view or modify. (See "Default Security Policies for MBeans" in the Oracle WebLogic Server MBean Reference.)

When you invoke the WLST connect command, you can supply user credentials by doing any of the following:

- Enter the credentials on the command line. This option is recommended only if you are using WLST in interactive mode.
  
  For example:
  ```
  connect('weblogic', 'welcome1', 'localhost:7001')
  ```

  For more information, see "connect" in WebLogic Scripting Tool Command Reference.

- Enter the credentials on the command line, then use the storeUserConfig command to create a user configuration file that contains your credentials in an encrypted form and a key file that WebLogic Server uses to unencrypt the credentials. On subsequent WLST sessions (or in WLST scripts), supply the name of the file instead of entering the credentials on the command line. This option is recommended if you use WLST in script mode because it prevents you from storing unencrypted user credentials in your scripts.

  For example, to create the user configuration file and key file:
  ```
  connect('weblogic', 'welcome1', 'localhost:7001')
  storeUserConfig('c:/myFiles/myuserconfigfile.secure',
                  'c:/myFiles/myuserkeyfile.secure')
  ```

  To use the user configuration file and key file:
connect(userConfigFile='c:/myfiles/myuserconfigfile.secure',
userKeyFile='c:/myfiles/myuserkeyfile.secure')

For more information, see "connect" and "storeUserConfig" in WebLogic Scripting Tool Command Reference.

- Invoke the connect command from a directory that contains the domain's boot.properties file. By default, when you create an Administration Server, WebLogic Server encrypts the credentials and stores them in a boot.properties file. WLST can use this file only if you start WLST from the domain directory.

For example, if you have not deleted the domain's boot.properties file, you can start WLST and invoke the connect command as follows:

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

For more information, see "connect" in WebLogic Scripting Tool Command Reference.

2.3.2.2 Writing and Reading Encrypted Configuration Values

Some attributes of a WebLogic domain’s configuration are encrypted to prevent unauthorized access to sensitive data. For example, the password that a JDBC data source uses to connect to an RDBMS is encrypted.

The attribute values are saved in the domain’s configuration document as an encrypted string. In a running server instance, the values are available as an MBean attribute in the form of an encrypted byte array. The names of encrypted attributes end with Encrypted. For example, the ServerMBean exposes the password that is used to secure access through the IIOP protocol in an attribute named DefaultIIOPPasswordEncrypted.

Oracle recommends the following pattern for writing and reading encrypted attributes:

With WLST offline:

- To write an encrypted value, pass the name of the encrypted attribute and an unencrypted string to the set command. For example:
  
  set('DefaultIIOPPasswordEncrypted', 'mypassword')

  WLST encrypts the string and writes the encrypted value to the domain’s configuration file.

  For more information, see "set" in WebLogic Scripting Tool Command Reference.

- WLST offline does not display the unencrypted value of an encrypted attribute. If you use the ls command to display management attributes, WLST offline returns asterisks as the value of encrypted attributes. If you use the get command, WLST offline returns a byte array that represents asterisks.

  For example:

  wls:/offline/wl_server/Server/examplesServer>ls()

  returns

  ... -rw- DefaultIIOPPasswordEncrypted ********
  ...
wls:/offline/wl_server/Server/examplesServer>get('DefaultIIOPPasswordEncrypted')

returns
array([42, 42, 42, 42, 42, 42, 42, 42], byte)

For more information, see "ls" and "get" in WebLogic Scripting Tool Command Reference.

With WLST online, for each encrypted attribute, an MBean also contains an unencrypted version. For example, ServerMBean contains an attribute named DefaultIIOPPasswordEncrypted which contains the encrypted value and an attribute named DefaultIIOPPassword, which contains the unencrypted version of the value.

To write and read encrypted values with WLST online:

- To write an encrypted value, start an edit session. Then do either of the following:
  - Pass the name of the unencrypted attribute and an unencrypted string to the set command. For example:
    ```
    set('DefaultIIOPassword', 'mypassword')
    ```
  - Pass the name of the encrypted attribute and an encrypted byte array to the set command. You can use the encrypt command to create the encrypted byte array (see "encrypt" in WebLogic Scripting Tool Command Reference). For example:
    ```
    set('DefaultIIOPasswordEncrypted', encrypt('mypassword'))
    ```

    **Note:** Do not pass an unencrypted string to the encrypted attribute. The encrypted attribute assumes that the value you pass to it is already encrypted.

When you activate the edit, WebLogic Server writes the encrypted value to the domain's configuration file.

- To read the encrypted value of the attribute, pass the name of the encrypted attribute to the get command. For example:
  ```
  get('DefaultIIOPasswordEncrypted')
  ```

  returns
  ```
  array([105, 114, 111, 110, 115, 116, 101, 101, 108], byte)
  ```

- To read the unencrypted value of the attribute, pass the name of the unencrypted attribute to the get command. For example:
  ```
  get('DefaultIIOPassword')
  ```

  returns
  ```
  mypassword
  ```

### 2.3.3 Securing Access to Security Data

The user names and passwords of WebLogic Server users, security groups, and security roles are not stored in a WebLogic domain's XML configuration documents. Instead, a WebLogic domain uses a separate software component called an **Authentication provider** to store, transport, and provide access to security data.
Authentication providers can use different types of systems to store security data. The Authentication provider that WebLogic Server installs uses an embedded LDAP server.

When you use WLST offline to create a domain template, WLST packages the Authentication provider's data store along with the rest of the domain documents. If you create a domain from the domain template, the new domain has an exact copy of the Authentication provider's data store from the domain template.

You cannot use WLST offline to modify the data in an Authentication provider's data store.

You can, however, use WLST online to interact with an Authentication provider and add, remove, or modify users, groups, and roles. For more information, see Section 6.3, "Managing Security Data (WLST Online)".

2.4 Main Steps for Using WLST in Interactive or Script Mode

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

- Section 2.4.1, "Invoking WLST"
- Section 2.4.2, "Exiting WLST"
- Section 2.4.3, "Syntax for WLST Commands"

2.4.1 Invoking WLST

You can invoke WLST in the following ways:

- Execute the appropriate shell script for your environment.
- Execute the java weblogic.WLST command.
- Run a WLST script.
- Execute the WebLogic Scripting Tool command from the Start menu (Windows only).

---

**Note:** If you notice that it takes a long time to create or update a domain using WLST on a UNIX or Linux operating system, set the CONFIG_JVM_ARGS environment variable to the following value to resolve this issue:

-Djava.security.egd=file:/dev/.urandom

See also Section 2.7, "Running WLST from Ant."

2.4.1.1 Invoking WLST Using Provided Shell Scripts

To invoke WLST using a shell script, execute the commands that are appropriate for your environment. Environment variables are automatically set when you invoke WLST this way.
Main Steps for Using WLST in Interactive or Script Mode

Using the WebLogic Scripting Tool

2-9

UNIX

cd $WL_HOME/common/bin
./wlst.sh

$WL_HOME is the path to your WebLogic Server home directory.

Windows

cd $WL_HOME\common\bin
wlst.cmd

$WL_HOME is the path to your WebLogic Server home directory.

2.4.1.2 Invoking WLST Using the java Command

To invoke WLST using the java command, you must first set up your environment for WLST. To set up your environment for WLST:

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

2. Add WebLogic Server classes to the CLASSPATH environment variable and $WL_HOME\server\bin to the PATH environment variable, where $WL_HOME refers to the top-level installation directory for WebLogic Server.

You can use the setWLSEnv script to set the required variables:

- Windows: $WL_HOME\server\bin\setWLSEnv.cmd
- UNIX: $WL_HOME/server/bin/setWLSEnv.sh

On UNIX operating systems, the setWLSEnv.sh command does not set the environment variables in all command shells. Oracle recommends that you execute this command using the Korn shell.

After setting up your environment, use the following syntax to invoke WLST. See Table 2–1 for a description of the WLST command options. See Table 2–2 for a description of the SSL arguments.

java

[ -Dweblogic.security.SSL.ignoreHostnameVerification=true
-Dweblogic.security.TrustKeyStore=DemoTrust ]
[ -Dweblogic.security.JavaStandardTrustKeyStorePassPhrase=password]  
[ -Dweblogic.security.CustomTrustKeyStoreFileName=filename 
-Dweblogic.security.TrustKeystoreType=jks ]
[ -Dweblogic.security.CustomTrustKeyStorePassPhrase=password]
[ -Dweblogic.security.SSL.hostnameVerifier=classname ]
weblogic.WLST

[ -loadProperties propertyFilename ]
[ -skipWLSTModuleScanning ]
[ [-i] filePath.py ]
Main Steps for Using WLST in Interactive or Script Mode

Table 2–1 Command Options for WLST

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-loadProperties filename</td>
<td>Use this option to load properties into the WLST session, where filename is the name of a file that contains name=value pairs.</td>
</tr>
<tr>
<td></td>
<td>You cannot use this option when you are importing WLST as a Jython module (see Section 2.8, &quot;Importing WLST as a Jython Module&quot;).</td>
</tr>
<tr>
<td></td>
<td>Instead of using this command-line option, you can use the loadproperties WLST command. See &quot;loadProperties&quot; in WebLogic Scripting Tool Command Reference.</td>
</tr>
<tr>
<td>-skipWLSModuleScanning</td>
<td>Use this option to reduce startup time by skipping package scanning and caching for WebLogic Server modules.</td>
</tr>
</tbody>
</table>

[-i] filePath.py

Use this option to run a WLST script, where filePath.py is an absolute or relative pathname for the script.

By default, WLST exits (stops the Java process) after it executes the script. Include -i to prevent WLST from exiting.

Note: If a WLST script named wlstProfile.py exists in the directory from which you invoke WLST or in user.home (the home directory of the operating system user account as determined by the JVM), WLST automatically runs the wlstProfile.py script; you do not need to specify the name of this WLST script file on the command-line.

Instead of using this command-line option, you can use the following command after you start WLST:

eexecfile('filePath.py')

Table 2–2 SSL Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>-Dweblogic.security.SSL.ignoreHostnameVerification=true</td>
<td>Use these system properties if you plan to connect WLST to a WebLogic Server instance through an SSL listen port, and if the server instance is using the demonstration SSL keys and certificates.</td>
</tr>
<tr>
<td>-Dweblogic.security.TrustKeyStore=DemoTrust</td>
<td>ignoreHostnameVerification disables host name verification. TrustKeyStore causes WLST to trust the CA certificates in the demonstration trust keystore (WL_HOME\server\lib\DemoTrust.jks). TrustKeyStore is required if the server instance to which you want to connect is using the demonstration identity and certificates. By default, WLST trusts only the CA certificates in the Java Standard Trust keystore (SDK_HOME\jre\lib\security\cacerts).</td>
</tr>
<tr>
<td>-Dweblogic.security.JavaStandardTrustKeyStorePassPhrase=password</td>
<td>Password that was used to secure the Java Standard Trust keystore. If the Java Standard Trust keystore is protected by a password, and if you want to trust its CA certificates, you must use this argument. By default, the Java Standard Trust keystore is not protected by a password.</td>
</tr>
<tr>
<td>-Dweblogic.security.CustomTrustKeyStoreFileName=filename -Dweblogic.security.TrustKeystoreType=jks</td>
<td>Causes WLST to trust the CA certificates in a custom keystore that is located at filename. You must use both arguments to trust custom keystores. The filename must match exactly the ServerMBean.CustomTrustKeyStoreFileName value persisted in config.xml; if the value specified in the CustomTrustKeyStoreFileName attribute is a relative pathname, you must also specify the same relative pathname in this argument.</td>
</tr>
</tbody>
</table>
Examples
To use WLST in script mode:

java weblogic.WLST
c:/Oracle/Middleware/wlserver/templates/scripts/wlst/distributeQueues.py

To run a WLST script on a WebLogic Server instance that uses the SSL listen port and the demonstration certificates:

java -Dweblogic.security.SSL.ignoreHostnameVerification=true
-Dweblogic.security.TrustKeyStore=DemoTrust weblogic.WLST
c:/Oracle/Middleware/wlserver/templates/scripts/wlst/distributeQueues.py

To use WLST in interactive mode:

java weblogic.WLST

To connect to a WebLogic Server instance after you start WLST in interactive mode:

wls:/offline> connect('weblogic','welcome1','localhost:7001')

2.4.1.3 Running Scripts
You can run WLST scripts in the following ways:

- Include the script in the java weblogic.WLST command. You can either include the full path to the script, as shown here:

  java weblogic.WLST
c:/Oracle/Middleware/wlserver/templates/scripts/wlst/distributeQueues.py

  or you can navigate to the directory where the script is located before invoking WLST, in which case you only need to include the script name in the command:

cd C:/Oracle/Middleware/wlserver_10.3/templates/scripts/wlst
java weblogic.WLST distributeQueues.py

- After invoking WLST, use the execfile() command:

  wls:/offline> execfile('c:/Oracle/Middleware/wlserver/templates/scripts/wlst/distributeQueues.py')

  Note: If you use backslashes (\) in the path name, be aware that WLST interprets them as a special character, which you must escape. For more information, see Section 2.4.3, "Syntax for WLST Commands."

Table 2–2 (Cont.) SSL Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>-Dweblogic.security.CustomTrustKeyStorePassPhrase=password</td>
<td>Password that was used to secure the custom keystore. You must use this argument only if the custom keystore is protected by a password.</td>
</tr>
<tr>
<td>-Dweblogic.security.SSL.hostnameVerifier=classname</td>
<td>Name of a custom Host Name Verifier class. The class must implement the weblogic.security.SSL.HostnameVerifier interface.</td>
</tr>
</tbody>
</table>
2.4.1.4 Invoking WLST From the Start Menu
On Windows, a shortcut on the Start menu sets the environment variables and invokes WLST (Oracle WebLogic > WebLogic Server > Tools > WebLogic Scripting Tool).

2.4.2 Exiting WLST
To exit WLST, enter the exit() command:

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

2.4.3 Syntax for WLST Commands
Follow this syntax when entering WLST commands or writing them in a script:

- Command names and arguments are case sensitive.
- Enclose arguments in single or double quotes. For example, ‘newServer’ or "newServer".
- If you specify a backslash character (\) in a string, either precede the backslash with another backslash or precede the entire string with a lower-case r character. The \ or r prevents Jython from interpreting the backslash as a special character.

For example when specifying a file path name that contains a backslash:

```
readTemplate('c:\\userdomains\\mytemplates\\mytemplate.jar')
```

or

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

---

**Note:** When specifying path names in WLST commands on a Windows machine, you can use a forward slash (/). For example:

```
readTemplate('c:/userdomains/templates/template.jar')
```

is as valid as

```
readTemplate('c:\\userdomains\\templates\\template.jar')
```

---

- When using WLST offline, the following characters are not valid in names of management objects: period (.), forward slash (/), or backward slash (\).
- If you need to cd to a management object whose name includes a forward slash (/), surround the object name in parentheses. For example:

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

2.5 Redirecting Error and Debug Output to a File
To redirect WLST information, error, and debug messages from standard out to a file, enter:

```
redirect(outputFile,[toStdOut])
stopRedirect()
```
This command also redirects the output of the dumpStack() and dumpVariables() commands.

For example, to redirect WLST output to the logs/wlst.log file under the directory from which you started WLST, enter the following command:

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

For more information, see "redirect" and "stopRedirect" in WebLogic Scripting Tool Command Reference.

### 2.6 Getting Help

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

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

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

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

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

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

The command returns the following:

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

**Syntax:**
disconnect()

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

### 2.7 Running WLST from Ant

WebLogic Server provides a custom Ant task, wlst, that invokes a WLST script from an Ant build file. You can create a WLST script (.py) file and then use this task to invoke the script file, or you can create a WLST script in a nested element within this task.

For more information about Ant, see the Apache Ant 1.7.1 Manual at http://ant.apache.org/manual/.

The wlst task is predefined in the version of Ant that is installed with WebLogic Server. To add this version of Ant to your build environment, run the following script:

```bash
WL_HOME\server\bin\setWLSEnv.cmd (or setWLSEnv.sh on UNIX)
```

where WL_HOME is the directory in which you installed WebLogic Server.

If you want to use the wlst task with your own Ant installation, include the following task definition in your build file:
<taskdef name="wlst"
        classname="weblogic.ant.taskdefs.management.WLSTTask" />

# 2.7.1 Parameters

Table 2–3 lists the \texttt{wlst} task parameters that you specify as attributes of the \texttt{<wlst>} element.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties=&quot;propsFile&quot;</td>
<td>Name and location of a properties file that contains name-value pairs that you can reference in your WLST script.</td>
<td>No</td>
</tr>
<tr>
<td>fileName=&quot;fileName&quot;</td>
<td>Name and location of the WLST script file that you would like to execute. If the specified WLST script file does not exist, this task fails.</td>
<td>Yes, if no nested \texttt{&lt;script&gt;} is used.</td>
</tr>
<tr>
<td>arguments=&quot;arglist&quot;</td>
<td>List of arguments to pass to the script. These arguments are accessible using the \texttt{sys.argv} variable.</td>
<td>No</td>
</tr>
<tr>
<td>failOnError=&quot;value&quot;</td>
<td>Boolean value specifying whether the Ant build will fail if this task fails.</td>
<td>No; default is true.</td>
</tr>
<tr>
<td>executeScriptBeforeFile=&quot;value&quot;</td>
<td>Boolean value specifying whether this task invokes the script in the nested \texttt{&lt;script&gt;} element before the script file specified by the \texttt{fileName} attribute. This attribute defaults to \texttt{true}, specifying that the embedded script is invoked first.</td>
<td>No; default is true.</td>
</tr>
<tr>
<td>debug=&quot;value&quot;</td>
<td>Boolean value specifying whether debug statements should be output when this task is executed.</td>
<td>No; default is false.</td>
</tr>
<tr>
<td>replaceProperties=&quot;value&quot;</td>
<td>Boolean value that specifies whether Ant property expansion will work in the specified WLST script.</td>
<td>No; default is true.</td>
</tr>
</tbody>
</table>

# 2.7.2 Parameters Specified as Nested Elements

The following sections describe the \texttt{wlst} task parameters that you specify as nested elements of the \texttt{<wlst>} element.

## 2.7.2.1 script

Contains a WLST script. This element is required if you do not use the \texttt{fileName} attribute to name a script file.

## 2.7.2.2 classpath

Specifies classes to add to the classpath. Use this element if your script requires classes that are not already on the classpath.

This element is the standard Ant \texttt{classpath} element. You can specify a reference to a \texttt{path} element that you have defined elsewhere in the build file or nest elements that specify the files and directories to add to the class path. See "Path-like Structures" in \textit{Apache Ant 1.7.1 Manual} at \url{http://ant.apache.org/manual/using.html#path}.

# 2.7.3 Examples

The following sections show examples for the \texttt{createServer}, \texttt{loop}, and \texttt{error} targets.
2.7.3.1 Example 1
In the following example, the createServer target does the following:

- Adds classes to the task's classpath.
- Executes the script in the nested script element. This script connects to a WebLogic domain's Administration Server at t3://localhost:7001. (Note that executeScriptBeforeFile is set to true, so this is invoked before the specified WLST script file.)
- Executes the script file myscript.py that is specified by the fileName attribute. The script file is located in the directory from which you started Ant. You could use such a file to start an edit session, create a new server, save, and activate the configuration changes.
- Defines three arguments that are passed to the script. These arguments are accessible using the sys.argv variable.
- Continues execution, as per the failOnError="false" setting, even if the wlst Ant task fails to execute.
- Disables debugging.

```xml
<target name="configServer">
  <wlst debug="false" failOnError="false" executeScriptBeforeFile="true" fileName="./myscript.py">
    <classpath>
      <pathelement location="${my.classpath.dir}"/>
    </classpath>
    <script>
      connect('weblogic','welcome1','t3://localhost:7001')
    </script>
  </wlst>
</target>
```

2.7.3.2 Example 2
In the following example, the loop target does the following:

- Adds classes to the task's classpath using a path reference.
- Executes the WLST script file myscript.py in the directory from which you started Ant. (Note that executeScriptBeforeFile is set to false, so the WLST script file is executed first, before the embedded script.)
- Executes the embedded script to connect to the server at t3://localhost:7001 and access and print the list of servers in the WebLogic domain.
- Results in a build failure if the wlst task fails to execute, as per the failOnError="true" setting.
- Enables debugging.

```xml
<path id="my.classpath">
  <pathelement location="${my.classpath.dir}"/>
</path>

<target name="loop">
  <wlst debug="true" executeScriptBeforeFile="false" fileName="./myscript.py" failOnError="true">
    <classpath>
      <pathelement location="${my.classpath.dir}"/>
    </classpath>
    <script replaceProperties="true">
      <!-- YourWLSTScriptHere goes here -->
    </script>
  </wlst>
</target>
```
import 'In the target loop'
connect('${admin.user}','${admin.password}','t3://localhost:7001')
svrs = cmo.getServer()
print 'Servers in the domain are'
for x in svrs: print x.getName()
</script>
</wlst>
</target>

2.7.3.3 Example 3
In the following example, the error target:

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

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

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

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

To import WLST as a Jython module:

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

2. Use the writeIniFile command to convert WLST definitions and method declarations to a .py file.
   wls://(offline)> writeIniFile("wl.py")
   The Ini file is successfully written to wl.py
   wls://(offline)>

3. Open a new command shell and invoke Jython directly by entering the following command:
   c:\>java org.python.util.jython
   The Jython package manager processes the JAR files in your classpath. The Jython prompt appears:
   >>>

4. Import the WLST module into your Jython module using the Jython import command.
5. Now you can use WLST methods in the module. For example, to connect WLST to a server instance:

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

---

**Notes:** When using WLST as a Jython module, in all WLST commands that have a block argument, block is always set to true, specifying that WLST will block user interaction until the command completes. See "WLST Command and Variable Reference" in *WebLogic Scripting Tool Command Reference*.

When running recorded scripts directly in a Jython interpreter, boolean values of `true` and `false` can cause an error. Before running recorded scripts in a Jython interpreter, make one of the following changes to the script:

- Add the following two lines to the script to define the true and false values:

  ```python
  true=1
  false=0
  ```

- Change `true` and `false` values to `wl.true` or `wl.false`, where *wl* is the name from the import.

---

### 2.9 Customizing WLST

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

---

**Note:** You can customize the default WLST home directory by passing the following argument on the command line:

```
-Dweblogic.wlstHome=<another-directory>
```

---

The following table describes ways to customize WLST.
You can customize WLST to include integrated help for any custom WSLT commands you've defined. To add integrated help, you define the help text for each command group and command in a resource bundle, which can be either a class or a property resource file. You can define the help for multiple command groups in the same resource bundle.

### Note:
The resource bundle must be present in the classpath.

The resource bundle contains the following entries for each command group:

```xml
<commandGroup>_ShortDescription=
<short description of command group>
<commandGroup>_Description=

\n<description of command group>
```

### Table 2-4  Customizing WLST

<table>
<thead>
<tr>
<th>To define custom...</th>
<th>Do the following...</th>
<th>For a sample script, see...</th>
</tr>
</thead>
<tbody>
<tr>
<td>WLST commands</td>
<td>Create a Python script defining the new commands and copy that file to <code>WL_HOME/common/wlst</code>.</td>
<td>Section 2.9.2, &quot;sample.py Sample Script&quot;</td>
</tr>
<tr>
<td>WLST commands</td>
<td>Create a Python script defining the new commands and copy that file to <code>WL_HOME/common/wlst/lib</code>.</td>
<td>Section 2.9.3, &quot;wlstLibSample.py Sample Script&quot;</td>
</tr>
<tr>
<td>The scripts located within this directory are imported as Jython libraries.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| WLST commands       | Create a Python script defining the new commands and copy that file to `WL_HOME/common/wlst/modules`. | `WL_HOME/common/wlst/modules/wlstModule.py`
| This script can be imported into other Jython modules, as described in Section 2.8, "Importing WLST as a Jython Module." | A JAR file, `jython-modules.jar`, which contains all of the Jython modules that are available in Jython 2.2.1, is also available within this directory. | |
| Integrated help for custom WLST commands | Define the help text for each command group and command in a resource bundle. | Not applicable. |
| Use `addHelpCommandGroup` to add a command group to the list of command groups that are displayed by the `help()` command. | | |
| Use `addHelpCommand` to add a command to the list of commands that is displayed by the `help('commandGroup')` command. | | |
| Typically, you will call the `addHelpCommandGroup` and `addHelpCommand` from the `.py` file that contains the definition of your custom commands. | | |
| See Section 2.9.1, "Adding Integrated Help for Custom Commands" for more information. | | |

### 2.9.1 Adding Integrated Help for Custom Commands

You can customize WLST to include integrated help for any custom WSLT commands you've defined. To add integrated help, you define the help text for each command group and command in a resource bundle, which can be either a class or a property resource file. You can define the help for multiple command groups in the same resource bundle.

### Table 2-4  Customizing WLST

<table>
<thead>
<tr>
<th>To define custom...</th>
<th>Do the following...</th>
<th>For a sample script, see...</th>
</tr>
</thead>
<tbody>
<tr>
<td>WLST commands</td>
<td>Create a Python script defining the new commands and copy that file to <code>WL_HOME/common/wlst</code>.</td>
<td>Section 2.9.2, &quot;sample.py Sample Script&quot;</td>
</tr>
<tr>
<td>WLST commands</td>
<td>Create a Python script defining the new commands and copy that file to <code>WL_HOME/common/wlst/lib</code>.</td>
<td>Section 2.9.3, &quot;wlstLibSample.py Sample Script&quot;</td>
</tr>
<tr>
<td>The scripts located within this directory are imported as Jython libraries.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| WLST commands       | Create a Python script defining the new commands and copy that file to `WL_HOME/common/wlst/modules`. | `WL_HOME/common/wlst/modules/wlstModule.py`
| This script can be imported into other Jython modules, as described in Section 2.8, "Importing WLST as a Jython Module." | A JAR file, `jython-modules.jar`, which contains all of the Jython modules that are available in Jython 2.2.1, is also available within this directory. | |
| Integrated help for custom WLST commands | Define the help text for each command group and command in a resource bundle. | Not applicable. |
| Use `addHelpCommandGroup` to add a command group to the list of command groups that are displayed by the `help()` command. | | |
| Use `addHelpCommand` to add a command to the list of commands that is displayed by the `help('commandGroup')` command. | | |
| Typically, you will call the `addHelpCommandGroup` and `addHelpCommand` from the `.py` file that contains the definition of your custom commands. | | |
| See Section 2.9.1, "Adding Integrated Help for Custom Commands" for more information. | | |
For example, the following lines define the short description and description for the command group navigate:

```java
navigate_ShortDescription=
   Lists commands for navigating the hierarchy of beans.
navigate_Description=
   \n   Navigates the hierarchy of beans and controls the prompt display. \n```

When you enter the `help()` command to display a list of command groups, the short description for `navigate` is displayed in the listing:

```shell
wls:/offline> help()
WLST is a command line scripting tool to configure and administer a WebLogic Server. Try:
   help('all')        List all WLST commands available.
   help('browser')    List commands for browsing the hierarchy.
   help('navigate')   List commands for navigating the bean hierarchy.
```

When you enter the `help('navigate')` command, the description is displayed above the list of commands in the group:

```shell
wls:/offline> help('navigate')
Navigates the hierarchy of beans and controls the prompt display.
   help('mycd')       Navigate the hierarchy of beans.
   help('myprmpt')   Toggle the display of path information at the prompt.
```

The resource bundle contains help text entries for commands using a standard pattern. For each command name, there are several entries:

```text
<commandName>_ShortDescription
<commandName>_Description
<commandName>_Example
<commandName>_Syntax
```

The following defines the help for `mycd` command:

```java
mycd_ShortDescription=
   Navigate the hierarchy of beans.
mycd_Description=
   \n   Navigate the hierarchy of configuration or runtime beans. This \n   command uses a model that is similar to navigating a file system \n   in a Windows or UNIX command shell. For example, to navigate back \n   to a parent configuration or runtime bean, enter mycd(".."). The \n   character string .. (dot-dot) refers to the directory immediately \n   above the current directory. To get back to the root configuration \n   after navigating to a configuration or runtime bean that is \n   deep in the hierarchy, enter mycd("/"). \n
mycd_Example=
   wls:/mydomain/serverConfig> mycd('Servers')
   wls:/mydomain/serverConfig/Servers> mycd('myserver')
   wls:/mydomain/serverConfig/Servers/myserver>mycd('../..')
   wls:/mydomain/serverConfig>

mycd_Syntax=
   mycd(mbeanName)
   \n   mbeanName = Path to the configuration or runtime bean in the namespace.\n```

The short description is shown to the right of the command name when you enter the `help('commandGroup')` command to list all commands in a group:
Customizing WLST

2.20 Oracle Fusion Middleware Oracle WebLogic Scripting Tool

2.9.2 Sample Script

This sample script demonstrates how to define a new WLST command.

```python
def wlstHomeSample():
    print 'Sample wlst home command'
```

Within this script, the `wlstHomeSample()` command is defined, which prints a String, as follows:

```python
wls:/offline> wlstHomeSample()
Sample wlst home command
```

2.9.3 WlstLibSample.py Sample Script

This sample script demonstrates usage of the WLST lib directory, where layered products and ISVs can add commands to WLST in their namespace. The script has the

```python
wls:/offline> help('navigate')
Navigates the hierarchy of beans and controls the prompt display.
help('mycd') Navigate the hierarchy of beans.
help('myprmpt') Toggle the display of path information at the prompt.
```

The description, example, and syntax are displayed when you enter the `help('commandName')` command:

```python
wls:/offline> help('mycd')
```

Description:

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

Syntax:

```
mycd(mbeanName)
```

- `mbeanName` = Path to the configuration or runtime bean in the namespace.

Example:

```python
wls:/mydomaina/serverConfig> mycd('Servers')
wls:/mydomain/serverConfig/Servers> mycd('myserver')
wls:/mydomain/serverConfig/Servers/myserver>mycd('../..')
wls:/mydomain/serverConfig>
```

After defining the help text in the resource bundle, use `addHelpCommandGroup` to add the command group name to the list of command groups output by the `help()` command. Use `addHelpCommand` to add each command in a group to the list of commands displayed by the `help('commandGroup')` command. See "addHelpCommandGroup" and "addHelpCommand" in the WebLogic Scripting Tool Command Reference.

For more information on resource bundles and localization, refer to http://java.sun.com/javase/6/docs/api/java/util/ResourceBundle.html.

For more information on resource bundles and localization, refer to http://java.sun.com/javase/6/docs/api/java/util/ResourceBundle.html.
same structure as the example in the previous section. It differs only in that it is stored
in the \texttt{WL\_HOME/common/wlst/lib} directory instead of the \texttt{WL\_HOME/common/wlst} directory.

\begin{verbatim}
def wlstExampleCmd()
    print 'Example command'
\end{verbatim}

Within this script, the \texttt{wlstExampleCmd()} command is defined, which prints a
String, as follows:

\begin{verbatim}
wlst:/(offline)>wlstLibSample.wlstExampleCmd()
Example command
\end{verbatim}
WLST enables you to create a new WebLogic domain or update an existing WebLogic domain without connecting to a running WebLogic Server (that is, using WLST offline)—supporting the same functionality as the Configuration Wizard.

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

- Section 3.1, "Creating and Using a Domain Template (Offline)"
- Section 3.2, "Alternative: Using the configToScript Command"
- Section 3.3, "Considerations for Clusters, JDBC, and JMS Resources"

For information about sample scripts that you can use to create WebLogic domains, see Section 1.4.2, "WLST Offline Sample Scripts".

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

3.1 Creating and Using a Domain Template (Offline)

A domain template is a JAR file that contains domain configuration documents, applications, security data, startup scripts, and other information needed to create a WebLogic domain. To create and use a domain template, perform the steps described in Table 3–1.

Note: If you notice that it takes a long time to create or update a domain using WLST, set the CONFIG_JVM_ARGS environment variable to the following value to resolve this issue:

-Djava.security.egd=file:/dev/.urandom

<table>
<thead>
<tr>
<th>To...</th>
<th>Use this command...</th>
<th>For more information, see...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open an existing WebLogic domain or template</td>
<td>readDomain(domainDirName) readTemplate(templateFileName)</td>
<td>“readDomain” and “readTemplate” in WebLogic Scripting Tool Command Reference</td>
</tr>
</tbody>
</table>
Creating and Using a Domain Template (Offline)

Table 3–1  (Cont.) Steps for Creating a Domain Template (Offline)

<table>
<thead>
<tr>
<th>To...</th>
<th>Use this command...</th>
<th>For more information, see...</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Optional) Modify the WebLogic domain</td>
<td>Browsing and editing commands</td>
<td>Section 3.1.1, &quot;Browsing Information About the Configuration Hierarchy (Offline)&quot; Section 3.1.2, &quot;Editing a WebLogic Domain (Offline)&quot;</td>
</tr>
<tr>
<td>Set the password for the default user, if it is not already set</td>
<td>cd('/Security/domainname/User/username')</td>
<td>Section 1.4.2, &quot;WLST Offline Sample Scripts&quot;</td>
</tr>
<tr>
<td>The default username and password must be set before you can write the domain template.</td>
<td>cmo.setPassword('password')</td>
<td></td>
</tr>
<tr>
<td>Write the domain configuration information to a domain template</td>
<td>writeTemplate(templateName)</td>
<td>&quot;writeTemplate&quot; in WebLogic Scripting Tool Command Reference</td>
</tr>
<tr>
<td>Use the template to create a domain</td>
<td>createDomain(domainTemplate, domainDir, user, password)</td>
<td>&quot;createDomain&quot; in WebLogic Scripting Tool Command Reference</td>
</tr>
<tr>
<td><strong>Note:</strong> The Configuration Wizard can also use the domain template. See Creating Domains Using the Configuration Wizard.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3.1.1 Browsing Information About the Configuration Hierarchy (Offline)

WLST offline provides read and write access to the configuration data that is persisted in the WebLogic domain’s config directory or in a domain template JAR created using Template Builder. This data is a collection of XML documents and expresses a hierarchy of management objects. The schemas that define a WebLogic domain’s configuration document are in the following locations:

- [http://xmlns.oracle.com/weblogic/domain/1.0/domain.xsd](http://xmlns.oracle.com/weblogic/domain/1.0/domain.xsd)
- [http://xmlns.oracle.com/weblogic/security/1.0/security.xsd](http://xmlns.oracle.com/weblogic/security/1.0/security.xsd)
- [http://xmlns.oracle.com/weblogic/weblogic-diagnostics/1.0/weblogic-diagnostics.xsd](http://xmlns.oracle.com/weblogic/weblogic-diagnostics/1.0/weblogic-diagnostics.xsd)
- In JAR files under `WL_HOME/server/lib/schema`, where `WL_HOME` is the directory in which you install WebLogic Server. Within this directory:
  - The domain.xsd document is represented in the weblogic-domain-binding.jar under the pathname META-INF/schemas/schema-0.xsd.
  - The security.xsd document is represented in the weblogic-domain-binding.jar under the pathname META-INF/schemas/schema-1.xsd.
  - The weblogic-diagnostics.xsd document is represented in the diagnostics-binding.jar under the pathname META-INF/schemas/schema-0.xsd.
WLST represents this hierarchy as a file system. The root of the file system is the management object that represents the WebLogic domain. Below the domain directory is a collection of directories for managed-object types; each instance of the type is a subdirectory under the type directory; and each management attribute and operation is a file within a directory. The name of an instance directory matches the value of the management object’s Name attribute. If the management object does not have a Name attribute, WLST generates a directory name using the following pattern: NO_NAME_number, where number starts at 0 (zero) and increments by 1 for each additional instance.

To navigate the hierarchy, you use such WLST commands as cd, ls, and pwd in a similar way that you would navigate a file system in a UNIX or Windows command shell (see Table 3–2).

---

**Note:** As a performance optimization, WebLogic Server does not store most of its default values in the domain’s configuration files. In some cases, this optimization prevents entire management objects from being displayed by WLST offline (because WebLogic Server has never written the corresponding XML elements to the domain’s configuration files). For example, if you never modify the default logging severity level for a WebLogic domain while the domain is active, WLST offline will not display the domain’s Log management object.

If you want to change the default value of attributes whose management object is not displayed by WLST offline, you must first use the create command to create the management object. Then you can cd to the management object and change the attribute value. See "create" in WebLogic Scripting Tool Command Reference.

---

### Table 3–2 Displaying WebLogic Domain Configuration Information (Offline)

<table>
<thead>
<tr>
<th>To...</th>
<th>Use this command...</th>
<th>For more information, see this section in WebLogic Scripting Tool Command Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Navigate the hierarchy of management objects</td>
<td>cd(path)</td>
<td>&quot;cd&quot;</td>
</tr>
<tr>
<td>List child attributes or management objects for the current management object</td>
<td>ls(['a'</td>
<td>'c'])</td>
</tr>
<tr>
<td>Toggle the display of the management object navigation path information at the prompt</td>
<td>prompt(['off'</td>
<td>'on'])</td>
</tr>
<tr>
<td>Display the current location in the configuration hierarchy</td>
<td>pwd()</td>
<td>&quot;pwd&quot;</td>
</tr>
<tr>
<td>Display all variables used by WLST</td>
<td>dumpVariables()</td>
<td>&quot;dumpVariables&quot;</td>
</tr>
<tr>
<td>Display the stack trace from the last exception that occurred while performing a WLST action</td>
<td>dumpStack()</td>
<td>&quot;dumpStack&quot;</td>
</tr>
</tbody>
</table>

---

### 3.1.2 Editing a WebLogic Domain (Offline)

To edit a WebLogic domain using WLST offline, you can perform any of the tasks defined in the following table.
Alternative: Using the configToScript Command

Note: If you notice that it takes a long time to create or update a domain using WLST, set the CONFIG_JVM_ARGS environment variable to the following value to resolve this issue:

-Djava.security.egd=file:/dev/.urandom

Table 3–3 Editing a WebLogic Domain

<table>
<thead>
<tr>
<th>To...</th>
<th>Use this command...</th>
<th>For more information, see this section in WebLogic Scripting Tool Command Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add an application to a WebLogic domain</td>
<td>addTemplate(templateFileName)</td>
<td>&quot;addTemplate&quot;</td>
</tr>
<tr>
<td>Assign resources to one or more destinations (such as assigning servers to clusters)</td>
<td>assign(sourceType, sourceName, destinationType, destinationName)</td>
<td>&quot;assign&quot;</td>
</tr>
<tr>
<td>Unassign resources</td>
<td>unassign(sourceType, sourceName, destinationType, destinationName)</td>
<td>&quot;unassign&quot;</td>
</tr>
<tr>
<td>Create and delete management objects</td>
<td>create(name, childMBeanType) delete(name, childMBeanType)</td>
<td>&quot;create&quot; &quot;delete&quot;</td>
</tr>
<tr>
<td>Get and set attribute values</td>
<td>get(attrName) set(attrName, value)</td>
<td>&quot;get&quot; &quot;set&quot;</td>
</tr>
<tr>
<td>Set configuration options</td>
<td>setOption(optionName, value)</td>
<td>&quot;setOption&quot;</td>
</tr>
<tr>
<td>Load SQL files into a database</td>
<td>loadDB(dbVersion, connectionPoolName)</td>
<td>&quot;loadDB&quot;</td>
</tr>
</tbody>
</table>

3.2 Alternative: Using the configToScript Command

WLST includes a command, configToScript, that reads an existing WebLogic domain and outputs a WLST script that can recreate the WebLogic domain. See "configToScript" in WebLogic Scripting Tool Command Reference.

Unlike creating and using a domain template, the configToScript command creates multiple files that must be used together. (A domain template is a single JAR file.) In addition, the script that the configToScript command creates:

- Can only be run by WLST.
  A domain template can be used by WLST or the Configuration Wizard.
- Requires a WebLogic Server instance to be running. If a server isn't running, the script starts one.
  WLST offline or the Configuration Wizard can use domain templates to create WebLogic domains without starting a server instance.
- Contains only references to applications and other resources. When you run the generated script, the applications and resources must be accessible to the WebLogic domain through the file system.
A domain template is a JAR file that contains all applications and resources needed to create a WebLogic domain. Because the domain template is self-contained, you can use it to create WebLogic domains on separate systems that do not share file systems.

### 3.3 Considerations for Clusters, JDBC, and JMS Resources

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

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

*WL_HOME*\samples\domains\wl_server\setExamplesEnv.cmd (on Windows)

*WL_HOME/samples/domains/wl_server/setExamplesEnv.sh* (on UNIX)
During its lifetime, a server can transition through a number of operational states, such as shutdown, starting, standby, admin, resuming, and running. For more information about the server life cycle, see "Understanding Server Life Cycle" in Managing Server Startup and Shutdown for Oracle WebLogic Server.

The following sections describe how to use WebLogic Scripting Tool (WLST) to manage and monitor the server life cycle:

- Section 4.1, "Using WLST and Node Manager to Manage Servers"
- Section 4.2, "Starting and Managing Servers Without Node Manager"

For information on other techniques for starting and stopping server instances, see "Starting and Stopping Servers" in Managing Server Startup and Shutdown for Oracle WebLogic Server.

### 4.1 Using WLST and Node Manager to Manage Servers

Node Manager is a utility that enables you to control the life cycles of multiple servers through a single WLST session and a single network connection. (It can also automatically restart servers after a failure.) For more information about Node Manager, see the Node Manager Administrator’s Guide for Oracle WebLogic Server.

You can use WLST to do the following with Node Manager:

- Start a Node Manager.
- Connect to a Node Manager, then use the Node Manager to start and stop servers on the Node Manager machine. See Figure 4–1.
A Node Manager process is not associated with a specific WebLogic domain but with a machine. You can use the same Node Manager process to control server instances in any WebLogic domain, as long as the server instances reside on the same machine as the Node Manager process.

For information about the commands that WLST can use while acting as a Node Manager client, see "Node Manager Commands" in WebLogic Scripting Tool Command Reference.

- Connect to an Administration Server, then use the Administration Server to start and stop servers in the domain. See Figure 4–2.

In this case, WLST is a client of the Administration Server, and the Administration Server uses one or more Node Managers to start Managed Servers.
For information about the life cycle commands that WLST can use while acting as an Administration Server client, see "Life Cycle Commands" in WebLogic Scripting Tool Command Reference.

### 4.1.1 Using Node Manager to Start Servers on a Machine

WLST can connect to a Node Manager that is running on any machine and start one or more WebLogic Server instances on the machine. A WebLogic domain's Administration Server does not need to be running for WLST and Node Manager to start a server instance using this technique.

To connect WLST to a Node Manager and start servers:

1. Configure Node Manager to start servers.

2. Start WLST.

3. Start Node Manager.
   Usually, as part of configuring Node Manager, you create a Windows service or a daemon that automatically starts Node Manager when the host computer starts. See "Running Node Manager as a Service" in the Node Manager Administrator’s Guide for Oracle WebLogic Server.
   If Node Manager is not already running, you can log on to the host computer and use WLST to start it:
   ```
   wls:/offline> startNodeManager()
   ```
   For more information about startNodeManager, see "startNodeManager" in WebLogic Scripting Tool Command Reference.

4. Connect WLST to a Node Manager by entering the nmConnect command.
   ```
   wls:/offline> nmConnect('username','password','nmHost','nmPort','domainName','domainDir','nmType')
   ```
   For example,
   ```
   nmConnect('weblogic', 'welcome1', 'localhost', '5556', 'mydomain','c:/bea/user_projects/domains/mydomain','SSL')
   Connecting to Node Manager ...
   Successfully connected to Node Manager.
   wls:/nm/mydomain>
   ```
   For detailed information about nmConnect command arguments, see "nmConnect" in WebLogic Scripting Tool Command Reference.

5. Use the nmStart command to start a server.
   ```
   wls:/nm/mydomain> nmStart('AdminServer')
   starting server AdminServer
   ...
   Server AdminServer started successfully
   wls:/nm/mydomain>
   ```

6. Monitor the status of the Administration Server by entering the nmServerStatus command.
   ```
   wls:/nm/mydomain> nmServerStatus('serverName')
   RUNNING
   ```
7. Stop the server by entering the nmKill command.

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

For more information about WLST Node Manager commands, see "Node Manager Commands" in WebLogic Scripting Tool Command Reference.

### 4.1.2 Using Node Manager to Start Managed Servers in a WebLogic Domain or Cluster

To start Managed Servers and clusters using Node Manager:

1. Configure Node Manager to start servers.
   

2. Start WLST.

3. Start Node Manager.

   Usually, as part of configuring Node Manager, you create a Windows service or a daemon that automatically starts Node Manager when the host computer starts. See "Running Node Manager as a Service" in the Node Manager Administrator’s Guide for Oracle WebLogic Server

   If Node Manager is not already running, you can log on to the host computer and use WLST to start it:

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

   For more information about startNodeManager, see "startNodeManager" in WebLogic Scripting Tool Command Reference.

4. Start an Administration Server.

5. Connect WLST to the Administration Server instance 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'.
   Warning: An insecure protocol was used to connect to the server.
   To ensure on-the-wire security, the SSL port or Admin port should be used instead.

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

   For detailed information about connect command arguments, see "connect" in WebLogic Scripting Tool Command Reference.

6. Do any of the following:

   - To start a Managed Server, enter the following command, where `managedServerName` is the name of the server.
     ```
     start('managedServerName', 'Server')
     ```
To start a cluster, enter the following command, where `clusterName` is the name of the cluster.

```bash
start('clusterName', 'Cluster')
```

For more information, see "start" in *WebLogic Scripting Tool Command Reference*.

## 4.2 Starting and Managing Servers Without Node Manager

The following sections describe starting and managing server state without using the Node Manager:

- **Section 4.2.1, "Starting an Administration Server Without Node Manager"**
- **Section 4.2.2, "Managing Server State Without Node Manager"**

If you do not use Node Manager, WLST cannot start Managed Servers. For information on other techniques for starting and stopping server instances, see "Starting and Stopping Servers" in *Managing Server Startup and Shutdown for Oracle WebLogic Server*.

### 4.2.1 Starting an Administration Server Without Node Manager

To start an Administration Server without using Node Manager:

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

   For more information, see Chapter 3, "Creating WebLogic Domains Using WLST Offline"

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 `MW_HOME\user_projects\domains\domain_name`, where `MW_HOME` is the top-level installation directory of Oracle WebLogic products.

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

   - `bin\setDomainEnv.cmd` (Windows)
   - `bin/setDomainEnv.sh` (UNIX: Oracle recommends that you run this script from the Korn shell.)

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

5. Invoke WLST by as described in Section 2.4.1, "Invoking WLST."

   The WLST prompt appears.

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

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

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

   For detailed information about `startServer` command arguments, see "startServer" in *WebLogic Scripting Tool Command Reference*.

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

4.2.2 Managing Server State Without Node Manager

WLST life cycle commands enable you to control the states through which a server instance transitions. See "Life Cycle Commands" in WebLogic Scripting Tool Command Reference. Oracle recommends that you enable and use the WebLogic domain's administration port when you connect to servers and issue administrative commands. See Section 2.3.1, "Securing the WLST Connection".

The commands in Example 4–1 explicitly move a server instance through the following server states: RUNNING->ADMIN->RUNNING->SHUTDOWN.

Start WebLogic Server before running this script.

Example 4–1  WLST Life Cycle Commands

# Connect to the Administration Server
connect("username","password","t3://localhost:7001")

# First enable the Administration Port. This is not a requirement.
# After you enable the Administration Port in a domain, WebLogic Server
# persists the setting in its configuration files. You do not need to repeat
# the process in future WLST sessions.
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")
The following sections describe how to navigate, interrogate, and edit MBeans using WLST:

- Section 5.1, "Navigating and Interrogating MBeans"
- Section 5.2, "Browsing Runtime MBeans"
- Section 5.3, "Navigating Among MBean Hierarchies"
- Section 5.4, "Finding MBeans and Attributes"
- Section 5.5, "Accessing Other WebLogic MBeans and Custom MBeans"

### 5.1 Navigating and Interrogating MBeans

WLST online provides simplified access to MBeans. While JMX APIs require you to use JMX object names to interrogate MBeans, WLST enables you to navigate a hierarchy of MBeans in a similar fashion to navigating a hierarchy of files in a file system.

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

In the configuration hierarchy, the root directory is `DomainMBean` (see "DomainMBean" in the Oracle WebLogic Server MBean Reference); the MBean type is a subdirectory under the root directory; each instance of the MBean type is a subdirectory under the MBean type directory; and MBean attributes and operations are nodes (like files) under the MBean instance directory. The name of the MBean instance directory matches the value of the MBean's `Name` attribute. If the MBean does not have a `Name` attribute, WLST generates a directory name using the following pattern: `NO_NAME_number`, where `number` starts at 0 (zero) and increments by 1 for each additional MBean instance.
**Figure 5–1 Configuration MBean Hierarchy**

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

WLST first connects to a WebLogic Server instance at the root of the server’s configuration MBeans, a single hierarchy whose root is DomainMBean. WLST commands provide access to all the WebLogic Server MBean hierarchies within a WebLogic domain, such as a server’s run-time MBeans, run-time MBeans for domain-wide services, and an editable copy of all the configuration MBeans in the domain. For more information, see “Tree Commands” in *WebLogic Scripting Tool Command Reference*.

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

### 5.1.1 Changing the Current Management Object

WLST online provides a variable, `cmo`, that represents the current management object. You can use this variable to perform any `get`, `set`, or `invoke` method on the management object. For example, the `cmo` variable enables the following command:

```plaintext
wls:/mydomain/edit> cmo.setAdministrationPort(9092)
```

The variable is available in all WLST hierarchies except custom and jndi.

WLST sets the value of `cmo` to the current WLST path. Each time you change directories, WLST resets the value of `cmo` to the current WLST path. For example, when you change to the serverRuntime hierarchy, `cmo` is set to ServerRuntime. When you change to the serverConfig hierarchy, `cmo` is set to DomainMBean. If you change to the Servers directory under DomainMBean, `cmo` is set to an instance of ServerMBean (see Example 5–1).

**Example 5–1 Changing the Current Management Object**

```plaintext
wls://(offline)> connect('username', 'password')
Connecting to weblogic server instance running at t3://localhost:7001 as username weblogic ...
Successfully connected to Admin Server 'myserver' that belongs to domain 'mydomain'.
Warning: An insecure protocol was used to connect to the server.
To ensure on-the-wire security, the SSL port or Admin port should be used instead.
wls:/mydomain/serverConfig> cmo
[MBeanServerInvocationHandler]com.bea:Name=mydomain,Type=Domain
wls:/mydomain/serverConfig> cd('Servers')
wls:/mydomain/serverConfig/Servers> cmo
[MBeanServerInvocationHandler]com.bea:Name=mydomain,Type=Domain
wls:/mydomain/serverConfig/Servers> cd('myserver')
```
Navigating and Interrogating MBeans

5.1.2 Navigating and Displaying Configuration MBeans Example

The commands in Example 5–2 instruct WLST to connect to an Administration Server instance and display attributes, operations, and child MBeans in DomainMBean.

**Example 5–2 Navigating and Displaying Configuration MBeans**

```
ws:/offline> connect('username','password')
ws:/mydomain/serverConfig> ls()
...
-r-- AdminServerName                              myserver
-r-- AdministrationMBeanAuditingEnabled           false
-r-- AdministrationPort                           9002
-r-- AdministrationPortEnabled                    false
-r-- AdministrationProtocol                       t3s
-r-- ArchiveConfigurationCount                    5
...
ws:/mydomain/serverConfig> cd('Servers')
ws:/mydomain/serverConfig/Servers> ls()
...
-r-- AcceptBacklog                                50
-r-- AdminReconnectIntervalSeconds                10
-r-- AdministrationPort                           0
-r-- AdministrationProtocol                       t3s
-r-- AutoKillIfFailed                             false
-r-- AutoMigrationEnabled                         false
-r-- AutoRestart                                  true
....
```
Navigating and Interrogating MBeans

```
$ wls:~/mydomain/serverConfig/Servers/myservice>
cd('Log/myservice')
$ wls:~/mydomain/serverConfig/Servers/myservice/Log/myservice>
ls()
```

```
-dr-- DomainLogBroadcastFilter
-dr-- LogFileFilter
-dr-- MemoryBufferFilter
-dr-- StdoutFilter

-r-- BufferSize 8
-r-- DateForatPattern MMM d, yyyy h:mm:ss a z
-r-- DomainLogBroadcastFilter null
-r-- DomainLogBroadcastSeverity Warning
-r-- DomainLogBroadcasterBufferSize 1
-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-- LoggerSeverity Info
-r-- LoggerSeverityProperties null
-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-- ServerLogBridgeUseParentLoggersEnabled false
-r-- StdoutFilter null
-r-- StdoutFormat standard
-r-- StdoutLogStack true
-r-- StdoutSeverity Warning
-r-- Type Log

-r-x freezeCurrentValue Void : String(attributeName)

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

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

**Note:** The read, write, and execute indicators assume that there are no restrictions to the current user's access privileges. A specific user might not be able to read values that WLST indicates as readable because the user might not have been given appropriate permission by the policies in the WebLogic Security realm. See "Default Security Policies for MBeans" in the *Oracle WebLogic Server MBean Reference*.

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

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

For more information, see “ServerRuntimeMBean” and "DomainRuntimeMBean" in the Oracle WebLogic Server MBean Reference.

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

Navigating and Displaying Runtime MBeans Example

The commands in Example 5–3 instruct WLST to connect to an Administration Server instance, navigate, and display server and domain run-time MBeans.

Example 5–3 Navigating and Displaying Runtime MBeans

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

```
dr--   ServerChannelRuntimes
dr--   ServerSecurityRuntime
dr--   ServerServices
dr--   ThreadPoolRuntime
dr--   WLDFAccessRuntime
dr--   WLDFRuntime
dr--   WTCRuntime
dr--   WorkManagerRuntimes

-r--   ActivationTime                                   1093958848908
-r--   AdminServer                                      true
-r--   AdminServerHost                                  
-r--   AdminServerListenPort                            7001
-r--   AdminServerListenPortSecure                      false
-r--   AdministrationPort                               
-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 Example 5–4 instruct WLST to navigate and display run-time MBeans on a Managed Server instance.

**Example 5–4  Navigating and Displaying Runtime MBeans on a Managed Server**

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

To navigate to a configuration MBean from the run-time 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 run-time MBean hierarchy to the configuration MBean hierarchy and back:

```
wlsh:/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')
wlsh:/mydomain/serverConfig> cd ('Servers/managed1')
wls:/mydomain/serverConfig/Servers/managed1> cd('Log/managed1')
wls:/mydomain/serverConfig/Servers/managed1/Log/managed1> serverRuntime()
```

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

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

For more information, see "Tree Commands" in WebLogic Scripting Tool Command Reference.

Alternatively, you can use the `currentTree` command to store your current MBean hierarchy location and to return to that location after navigating away from it. See "currentTree" in WebLogic Scripting Tool Command Reference.

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()
```
5.4 Finding MBeans and Attributes

To locate a particular MBean and attribute, you use the `find` command. WLST returns the pathname to the MBean that stores the attribute and its value. You can use the `getMBean` command to return the MBean specified by the path. For more information, see "find" and "getMBean" in WebLogic Scripting Tool Command Reference.

For example:

```plaintext
wls:/mydomain/edit !> find('DebugEjbCaching')
finding 'DebugEjbCaching' in all registered MBean instances ...
/Servers/AdminServer/ServerDebug/AdminServer                          false
/Servers/managed2/ServerDebug/managed2                           false
wls:/mydomain/edit !> bean=getMBean('Servers/managed2/ServerDebug/managed2')
wls:/mydomain/edit !> print bean
[MBeanServerInvocationHandler]bea.com:Name=managed2,Type=ServerDebug, Server=managed2
```

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

Alternatively, the `getPath` command returns the MBean path for a specified MBean instance or `ObjectName` for the MBean in the current MBean hierarchy. See "getPath" in WebLogic Scripting Tool Command Reference.

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

5.5 Accessing Other WebLogic MBeans and Custom MBeans

In addition to accessing WebLogic Server MBeans, WLST can access MBeans that WebLogic Integration and WebLogic Portal provide. It can also access MBeans that you create and register (custom MBeans) to configure or monitor your own resources. (For information on creating and registering your own MBeans, see "Instrumenting and Registering Custom MBeans" in Developing Manageable Applications With JMX for Oracle WebLogic Server.)

To navigate other WebLogic MBeans or custom MBeans, enter the `custom` command or the `domainCustom` command, depending on the MBean server (Runtime or Domain Runtime) on which the custom MBean is registered. You can use `custom` when WLST is connected to an Administration Server or a Managed Server instance. You can use `domainCustom` only when WLST is connected to an Administration Server. See Section 5.5.1, "Accessing Custom MBeans in the Domain Runtime MBean Server," for information about `domainCustom`.

WLST treats all non-WebLogic Server MBeans as custom MBeans:

- Instead of arranging custom MBeans in a hierarchy, WLST organizes and lists custom MBeans by JMX object name. All MBeans with the same JMX domain name are listed in the same WLST directory. For example, if you register all of
your custom MBeans with JMX object names that start with mycompany:, then WLST arranges all of your MBeans in a directory named mycompany.

- Custom MBeans cannot use the cmo variable because a stub is not available.
- Custom MBeans are editable, but not subject to the WebLogic Server change management process. You can use MBean get, set, invoke, and create and delete commands on them without first entering the startEdit command. See Section 6.1, "Using WLST Online to Update an Existing WebLogic Domain".

Here is an example of navigating custom MBeans on the Runtime MBean Server:

```wls
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- mycompany
drw- anothercompany
wls:/mydomain/custom> cd("mycompany")
wls:/mydomain/custom/mycompany> ls()
drw- mycompany:y1=x
drw- mycompany:y2=x
wls:/mydomain/custom/mycompany> cd("mycompany:y1=x")
wls:/mydomain/custom/mycompany/mycompany:y1=x> ls()
-prw- MyAttribute 10
wls:/mydomain/custom/mycompany/mycompany:y1=x>
```

### 5.5.1 Accessing Custom MBeans in the Domain Runtime MBean Server

Use the domainCustom() command to browse and invoke methods or perform operations on custom MBeans that are registered in the Domain Runtime MBean Server. This is similar to using the custom() command to access custom MBeans that are registered in the Runtime MBean Server, as described in Section 5.5. You can use the domainCustom() command only when WLST is connected to the Administration Server.

For information on using domainCustom(), see "domainCustom" in the WebLogic Scripting Tool Command Reference.

For information on how to access custom MBeans in the Domain Runtime MBean server, see "Make Local Connections to the Domain Runtime MBean Server" in Developing Custom Management Utilities With JMX for Oracle WebLogic Server. For information on creating and registering your own MBeans, see "Instrumenting and Registering Custom MBeans" in Developing Manageable Applications With JMX for Oracle WebLogic Server.
The following sections describe using WLST to update an existing WebLogic domain:

- Section 6.1, "Using WLST Online to Update an Existing WebLogic Domain"
- Section 6.2, "Using WLST Offline to Update an Existing WebLogic Domain"
- Section 6.3, "Managing Security Data (WLST Online)"
- Section 6.4, "Deploying Applications"

### 6.1 Using WLST Online to Update an Existing WebLogic Domain

Because WLST online interacts with an active WebLogic domain, all online changes to a domain are controlled by the change management process, which loosely resembles a database transaction. For more information on making and managing configuration changes, see "Configuration Change Management Process" in *Understanding Domain Configuration for Oracle WebLogic Server*.

*Table 6–1* describes the steps for using WLST online to update an existing WebLogic domain.

<table>
<thead>
<tr>
<th>To...</th>
<th>Use this command...</th>
<th>For more information, see this section in WebLogic Scripting Tool Command Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access the edit MBean hierarchy</td>
<td>edit()</td>
<td>&quot;edit&quot;</td>
</tr>
<tr>
<td></td>
<td>This command places WLST at the root of the edit MBean hierarchy, which is the editable DomainMBean.</td>
<td></td>
</tr>
<tr>
<td>Obtain a lock on the current configuration</td>
<td>startEdit([waitTimeInMillis], [timeoutInMillis], [exclusive])</td>
<td>&quot;startEdit&quot;</td>
</tr>
<tr>
<td>To indicate that configuration changes are in process, an exclamation point (!) appears at the end of the WLST command prompt.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modify the WebLogic domain</td>
<td>Browsing and online editing commands</td>
<td>&quot;Browse Commands&quot;</td>
</tr>
<tr>
<td>(Optional) Validate your edits</td>
<td>validate()</td>
<td>&quot;validate&quot;</td>
</tr>
</tbody>
</table>

*Table 6–1*  Steps for Updating an Existing WebLogic Domain (Online)
Using WLST Online to Update an Existing WebLogic Domain

The WLST online script in Example 6–1 connects WLST to an Administration Server, initiates an edit session that creates a Managed Server, saves and activates the change, initiates another edit session, creates a startup class, and targets it to the newly created server.

The interactive edit session in Example 6–2 changes an Administration Server running in development mode to production mode.

**Example 6–1 Creating a Managed Server**

```java
connect("username","password")
edit()
startEdit()
svr = cmo.createServer("managedServer")
svr.setListenPort(8001)
svr.setListenAddress("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")
exit()
```

**Example 6–2 Changing to Production Mode**

```java
wls:/offline> connect('username','password')
wls:/mydomain/serverConfig> edit()
Starting an edit session ...
```

### Table 6–1 (Cont.) Steps for Updating an Existing WebLogic Domain (Online)

<table>
<thead>
<tr>
<th>To...</th>
<th>Use this command...</th>
<th>For more information, see this section in WebLogic Scripting Tool Command Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save your changes</td>
<td>save()</td>
<td>&quot;save&quot;</td>
</tr>
<tr>
<td>Distribute your changes to the working configuration MBeans on all servers in the WebLogic domain</td>
<td>activate([timeout], [block])</td>
<td>&quot;activate&quot;</td>
</tr>
<tr>
<td>Release your lock on the configuration</td>
<td>stopEdit([defaultAnswer])</td>
<td>&quot;stopEdit&quot;</td>
</tr>
<tr>
<td>(Optional) Determine if a change you made to an MBean attribute requires you to re-start servers</td>
<td>isRestartRequired([attributeName])</td>
<td>&quot;isRestartRequired&quot;</td>
</tr>
</tbody>
</table>

For more information, see this section in WebLogic Scripting Tool Command Reference.
Started edit session, please be sure to save and activate your changes once you are done.

```
wls:/mydomain/edit !> cmo.setProductionModeEnabled(true)
wls:/mydomain/edit !> activate()
```

Activating all your changes, this may take a while ...
The edit lock associated with this edit session is released once the activation is completed.
The following non-dynamic attribute(s) have been changed on MBeans that require server re-start:

- **MBean Changed**: com.bea:Name=AdminServer,Type=WebServerLog,Server=AdminServer, WebServer=AdminServer  
  Attributes changed: RotateLogOnStartup
- **MBean Changed**: com.bea:Name=AdminServer,Type=WebServerLog,Server=AdminServer, WebServer=AdminServer  
  Attributes changed: RotateLogOnStartup
- **MBean Changed**: com.bea:Name=Domain1,Type=Log  
  Attributes changed: RotateLogOnStartup

Activation completed

```
wls:/mydomain/edit> exit()
```

### 6.1.1 Tracking Configuration Changes

For all changes that are initiated by WLST, you can use the `showChanges` command which displays all the changes that you made to the current configuration from the start of the WLST edit session, including any MBean operations that were implicitly performed by the server. See Example 6-3.

**Example 6-3  Displaying Changes**

```
connect('username','password')  
wls:/mydomain/serverConfig> edit()  
wls:/mydomain/edit> startEdit()  
Starting an edit session ...
Started edit session, please be sure to save and activate your changes once you are done.
#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 !> cmo.setListenAddress("localhost")  
wls:/mydomain/edit>Servers/managed2 !> showChanges()  
Changes that are in memory and saved to disc but not yet activated are:  
All changes that are made but not yet activated are:

- **MBean Changed**: com.bea:Name=Len,Type=Domain  
  Operation Invoked: create
  Attribute Modified: Servers
  Attributes Old Value: null  
  Attributes New Value: managed2
  Server Restart Required: false

- **MBean Changed**: com.bea:Name=managed2,Type=Server  
  Operation Invoked: modify
  Attribute Modified: ListenPort
  Attributes Old Value: null
  Attributes New Value: 7702
  Server Restart Required: false
The WLST online script in Example 6–4 connects WLST to a running 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.

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

**Example 6–4 Checking the Activation Task**

```python
at = getActivationTask()
changes = at.getChanges()
newstate = at.getState()
print 'The user for this Task is '+at.getUser()+ 'and the state is:'
print newstate
print 'The changes are:'
print changes
```

6.1.2 Undoing or Canceling Changes

WLST offers two commands to undo or cancel changes:

- The `undo` command reverts all unsaved or unactivated edits.
  
  You specify whether to revert all unactivated edits (including those that have been saved to disk), or all edits made since the last `save` operation. See "undo" in WebLogic Scripting Tool Command Reference.

- The `cancelEdit` command releases the edit lock and discards all unsaved changes. See "cancelEdit" in WebLogic Scripting Tool Command Reference.

6.1.3 Additional Operations and Attributes for Change Management

The standard change-management commands described in the previous section are convenience commands for invoking operations in the `ConfigurationManagerMBean`. In addition to these operations, the `ConfigurationManagerMBean` contains attributes and operations that describe edit sessions. For detailed information, see "ConfigurationManagerMBean" in the Oracle WebLogic Server MBean Reference.

To access this MBean, use the WLST `getConfigManager` command. See "getConfigManager" in WebLogic Scripting Tool Command Reference.

The WLST online script in Example 6–5 connects WLST to a server instance as an administrator (weblogic), checks if the current editor making changes is not the administrator, then cancels the configuration edits. The script also purges all the
completed activation tasks. You can use this script to make a fresh start to edit changes, but you should verify that the changes made by other editors are not needed.

Example 6–5  Using the Configuration Manager

```python
cconnect('weblogic','welcome1')
user = cmgr.getCurrentEditor()
if user != 'weblogic':
    cmgr.undo()
    cmgr.cancelEdit()
    cmgr.purgeCompletedActivationTasks()
```

6.2 Using WLST Offline to Update an Existing WebLogic Domain

To update an existing WebLogic domain using WLST offline, perform the steps described in Table 6–2.

Note: Oracle recommends that you do not use WLST offline to manage the configuration of an active WebLogic domain. Offline edits are ignored by running servers and can be overwritten by JMX clients such as WLST online or the WebLogic Server Administration Console.

Table 6–2  Steps for Updating an Existing WebLogic Domain (Offline)

<table>
<thead>
<tr>
<th>To...</th>
<th>Use this command...</th>
<th>For more information, see...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open an existing WebLogic domain for update</td>
<td>readDomain(domainDirName)</td>
<td>&quot;readDomain&quot; in WebLogic Scripting Tool Command Reference</td>
</tr>
<tr>
<td>Extend the current WebLogic domain (optional)</td>
<td>addTemplate(templateFileName)</td>
<td>&quot;addTemplate&quot; in WebLogic Scripting Tool Command Reference</td>
</tr>
<tr>
<td>Modify the WebLogic domain (optional)</td>
<td>Browsing and editing commands</td>
<td>Section 3.1.1, &quot;Browsing Information About the Configuration Hierarchy (Offline)&quot; Section 3.1.2, &quot;Editing a WebLogic Domain (Offline)&quot;</td>
</tr>
<tr>
<td>Save the WebLogic domain</td>
<td>updateDomain()</td>
<td>&quot;updateDomain&quot; in WebLogic Scripting Tool Command Reference</td>
</tr>
<tr>
<td>Close the WebLogic domain</td>
<td>closeDomain()</td>
<td>&quot;closeDomain&quot; in WebLogic Scripting Tool Command Reference</td>
</tr>
</tbody>
</table>

6.3 Managing Security Data (WLST Online)

In the WebLogic Security Service, an Authentication provider is the software component that proves the identity of users or system processes. An Authentication provider also remembers, transports, and makes that identity information available to various components of a system when needed.

A security realm can use different types of Authentication providers to manage different sets of users and groups. (See "Authentication Providers" in Developing Security Providers for Oracle WebLogic Server. You can use WLST to invoke operations on the following types of Authentication providers:
The default WebLogic Server Authentication provider, AuthenticatorMBean. By default, all security realms use this Authentication provider to manage users and groups.

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

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

- Section 6.3.1, "Determining If You Need to Access the Edit Hierarchy"
- Section 6.3.2, "Creating a User"
- Section 6.3.3, "Adding a User to a Group"
- Section 6.3.4, "Verifying Whether a User Is a Member of a Group"
- Section 6.3.5, "Listing Groups to Which a User Belongs"
- Section 6.3.6, "Listing Users and Groups in a Security Realm"
- Section 6.3.7, "Changing a Password"
- Section 6.3.8, "Protecting User Accounts in a Security Realm"
- Section 6.3.9, "Configuring Additional LDAP Authentication Providers"

For information about additional tasks that the AuthenticationProvider MBeans support, see "AuthenticationProviderMBean" in the Oracle WebLogic Server MBean Reference.

6.3.1 Determining If You Need to Access the Edit Hierarchy

If you are using WLST to change the configuration of a security MBean, you must access the edit hierarchy and start an edit session. For example, if you change the value of the LockoutThreshold attribute in UserLockoutManagerMBean, you must be in the edit hierarchy.

If you invoke security provider operations to add, modify, or remove data in a security provider data store, WLST does not allow you to be in the edit hierarchy. Instead, invoke these commands from the serverConfig or domainConfig hierarchy. For example, you cannot invoke the createUser operation in an AuthenticatorMBean MBean from the edit hierarchy. WLST enforces this restriction to prevent the possibility of incompatible changes. For example, an edit session could contain an unactivated change that removes a security feature and will invalidate modifications to the provider's data.

6.3.2 Creating a User

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

The method requires three input parameters. The password must be at least eight characters, with one special character or numeric character.

username password user-description
WLST cannot invoke this command from the edit hierarchy, but it can invoke the command from the serverConfig or domainConfig hierarchy.

The following WLST online script invokes `createUser` on the default authentication provider.

**Example 6–6 Creating a User**

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

print 'Creating a user ...'
atnr=cmo.getSecurityConfiguration().getDefaultRealm().lookupAuthenticationProvider('DefaultAuthenticator')
atnr.createUser('new_user','welcome1','new_admin')
print 'Created user successfully'
```

### 6.3.3 Adding a User to a Group

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

The method requires two input parameters:

- **groupname username**

WLST cannot invoke this command from the edit hierarchy, but it can invoke the command from the serverConfig or domainConfig hierarchy.

The following WLST online script invokes `addMemberToGroup` on the default Authentication Provider. For information on how to run this script, see Section 2.4.1, "Invoking WLST".

**Example 6–7 Adding a User to a Group**

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

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

### 6.3.4 Verifying Whether a User Is a Member of a Group

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

The method requires three input parameters:

- **groupname username boolean**

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

WLST cannot invoke this command from the edit hierarchy, but it can invoke the command from the serverConfig or domainConfig hierarchy.
The following WLST online script invokes `isMember` on the default Authentication Provider. For information on how to run this script, see Section 2.4.1, "Invoking WLST".

**Example 6–8 Verifying Whether a User is a Member of a Group**

```python
from weblogic.management.security.authentication import GroupEditorMBean
user = "my_user"
print "Checking if " + user + " is a Member of a group ... 
atnr=cmo.getSecurityConfiguration().getDefaultRealm().lookupAuthenticationProvider("DefaultAuthenticator")
if atnr.isMember('Administrators',user,True) == 0:
    print user+ " is not member of Administrators"
else:
    print user+ " is a member of Administrators"
```

### 6.3.5 Listing Groups to Which a User Belongs

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

The method requires one input parameter:

`memberUserOrGroupName`

where `memberUserOrGroupName` specifies the name of an existing user or a group. WLST cannot invoke this command from the edit hierarchy, but it can invoke the command from the serverConfig or domainConfig hierarchy.

The following WLST online script invokes `listMemberGroups` on the default Authentication provider. For information on how to run this script, see Section 2.4.1, "Invoking WLST".

**Example 6–9 Listing Groups to Which a User Belongs**

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

print "Listing the member groups ...

atnr=cmo.getSecurityConfiguration().getDefaultRealm().lookupAuthenticationProvider("DefaultAuthenticator")
x = atnr.listMemberGroups('my_user')
print x
```

The method returns a cursor value (for example, Cursor_16), which refers to a list of names. The `NameLister.haveCurrent`, `getCurrentName`, and `advance` operations iterate through the returned list and retrieve the name to which the current cursor position refers. See "NameListerMBean" in the *Oracle WebLogic Server MBean Reference*.

### 6.3.6 Listing Users and Groups in a Security Realm

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

- The `GroupReaderMBean.listGroups` and `UserReaderMBean.listUsers` methods take two input parameters: a pattern of user or group names to search for, and the maximum number of names that you want to retrieve.
Because a security realm can contain thousands (or more) of user and group names that match the pattern, the methods return a cursor, which refers to a list of names.

For more information, see the "listGroups" operation in the GroupReaderMBean and the "listUsers" operation in the UserReaderMBean in the Oracle WebLogic Server MBean Reference.

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

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

WLST cannot invoke these commands from the edit hierarchy, but it can invoke them from the serverConfig or domainConfig hierarchy.

The WLST online script in Example 6–10 lists all the users in a realm and the groups to which they belong. For information on how to run this script, see Section 2.4.1, "Invoking WLST".

**Example 6–10  Listing Users and Groups**

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

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

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

### 6.3.7 Changing a Password

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

WLST cannot invoke this command from the edit hierarchy, but it can invoke the command from the serverConfig or domainConfig hierarchy.
The following WLST online script invokes `changeUserPassword` on the default Authentication Provider: For information on how to run this script, see Section 2.4.1, "Invoking WLST".

**Example 6–11  Changing a Password**

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

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

### 6.3.8 Protecting User Accounts in a Security Realm

The `UserLockoutManagerMBean` provides a set of attributes to protect user accounts from intruders. By default, these attributes are set for maximum protection. You can decrease the level of protection for user accounts. For example, you can set whether or not lockout is enabled, increase the time period in which invalid login attempts are made before locking the user account, or change the amount of time a user account is locked.

The `UserLockoutManagerRuntimeMBean` provides a set of attributes for collecting lockout statistics, and operations for managing user lockouts. For example, you can get the number of users currently locked out, get the number of invalid login attempts since the server was started, or clear the lockout on a user account.

For more information about lockout configuration, see the "UserLockoutManagerMBean" interface in the Oracle WebLogic Server MBean Reference. For information about collecting lockout statistics and performing lockout operations, see the "UserLockoutManagerRuntimeMBean" interface in the Oracle WebLogic Server MBean Reference.

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

- **Section 6.3.8.1, "Set Consecutive Invalid Login Attempts"
- **Section 6.3.8.2, "Unlock a User Account"

Note that because these tasks edit MBean attributes, WLST must connect to the Administration Server, navigate to the edit hierarchy, and start an edit session.

#### 6.3.8.1 Set Consecutive Invalid Login Attempts

The following WLST online script sets the number of consecutive invalid login attempts before a user account is locked out. For information on how to run this script, see Section 2.4.1, "Invoking WLST".

**Example 6–12  Setting Consecutive Invalid Login Attempts**

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

edit()
startEdit()

# You have two choices for getting a user lockout manager to configure
# 1 - to configure the default realm's UserLockoutManager:
ulm=cmo.getSecurityConfiguration().getDefaultRealm().getUserLockoutManager()
```
# 2 - to configure another realm's UserLockoutManager:
#ulm=cmo.getSecurityConfiguration().lookupRealm("anotherRealm").getUserLockoutManager()

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

6.3.8.2 Unlock a User Account

The following WLST online script unlocks a user account. For information on how to
run this script, see Section 2.4.1, "Invoking WLST".

**Example 6–13 Unlocking a User Account**

```python
from weblogic.management.runtime import UserLockoutManagerRuntimeMBean
serverRuntime()
ulm=cmo.getServerSecurityRuntime().getDefaultRealmRuntime().getUserLockoutManagerRuntime()
#note1 : You can only manage user lockouts for the default realm starting from
#when the server was booted (versus other non-active realms).
#note2 : If the default realm's user lockout manager's LockoutEnabled attribute
#is false, then the user lockout manager's runtime MBean will be null.
#That is, you can only manage user lockouts in the default realm if its user
#lockout manager is enabled.
if ulm != None:
    ulm.clearLockout('myuser')
```

6.3.9 Configuring Additional LDAP Authentication Providers

In some cases, such as when installing some Oracle Fusion Middleware products, you
must add an additional external LDAP authentication provider to the WebLogic Server
security providers. This can be done either by using the WebLogic Server
Administration Console (see "Configure Authentication and Identity Assertion
Providers") or by using WLST.

**Example 6–14 shows how to use WLST to add an Oracle Internet Directory (OID)
authentication provider. To add other types of LDAP authentication providers,
substitute the appropriate class type in the createAuthenticationProvider
command, as shown in Table 6–3.**

---

**Note:** For important information about switching LDAP
authentication providers if the corresponding LDAP server will
contain the user or users who start the domain, see "Requirements for
Using an LDAP Authentication Provider" in *Securing Oracle WebLogic Server*.

---

**Example 6–14 Adding an Authentication Provider**

```python
connect ('adminUser', 'adminPassword', 't3://'+adminServerHost+':''+adminServerPort)
edit()
startEdit()
cd('/SecurityConfiguration/'+'+domainName'+'/Realms/myrealm')
# In the following command, substitute the appropriate class type
cmo.createAuthenticationProvider(LDAPProviderName,
  'weblogic.security.providers.authentication.OracleInternetDirectoryAuthenticator')
cd('/SecurityConfiguration/'+'+domainName'+'/Realms/myrealm/AuthenticationProviders')```
deploy("examplesWebApp","C:/Oracle/Middleware/wlserver_10.3/samples/server/examples/build/examplesWebApp")
Note: Please note the following when using WLST online to deploy applications:

- Deployment operations must be performed through the Administration Server. Your WLST client must connect to the Administration Server to invoke deployment commands.
- You do not need to be in an edit session to deploy applications.

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

6.4.2 Using WLST Offline to Deploy Applications

Table 6–4 describes the steps for using WLST offline to deploy applications in an existing domain.

<table>
<thead>
<tr>
<th>To...</th>
<th>Use this command...</th>
<th>For more information, see ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use the Template Builder to create an</td>
<td>not applicable</td>
<td>Creating Domain Templates Using the Domain Template</td>
</tr>
<tr>
<td>application template.</td>
<td></td>
<td>Builder</td>
</tr>
<tr>
<td>Open an existing WebLogic domain or</td>
<td>readDomain(domainDirName)</td>
<td>&quot;readDomain&quot; and &quot;readTemplate&quot; in WebLogic</td>
</tr>
<tr>
<td>template</td>
<td></td>
<td>Scripting Tool Command Reference</td>
</tr>
<tr>
<td>Add the application template to the</td>
<td>addTemplate(templateFileName)</td>
<td>&quot;addTemplate&quot; in WebLogic Scripting Tool Command</td>
</tr>
<tr>
<td>WebLogic domain</td>
<td></td>
<td>Reference</td>
</tr>
<tr>
<td>Save the WebLogic domain</td>
<td>updateDomain()</td>
<td>&quot;updateDomain&quot; in WebLogic Scripting Tool Command</td>
</tr>
<tr>
<td>Close the WebLogic domain</td>
<td>closeDomain()</td>
<td>&quot;closeDomain&quot; in WebLogic Scripting Tool Command</td>
</tr>
</tbody>
</table>

For an example of using the addTemplate command, see the following sample WLST script:

WL_HOME\common\templates\scripts\wlst\clusterMedRecDomain.py, where WL_HOME refers to the top-level installation directory for WebLogic Server.
You can use WLST to retrieve and update an application's deployment plan. When using WLST to update an application's deployment plan, you define variable definitions and variable assignments. A variable definition identifies a new value; a variable assignment associates the new value with the descriptor entity to be changed.

The following procedure describes how to use WLST in interactive mode. For information about using WLST in script or embedded mode, see Chapter 2, "Using the WebLogic Scripting Tool".

To update a deployment plan using WLST in interactive mode, perform the following steps:

1. Create a deployment plan for the application.
   
   For more information, see "Create a deployment plan" in the Oracle WebLogic Server Administration Console Help.

2. Start WLST in interactive mode.

3. Enter the following command to load the application and deployment plan. For example:

   ```
   plan=loadApplication(loadApplication('c:/Oracle/Middleware/user_projects/applications/mydomain/modules/medrec/assembly/target/medrec.ear', 'c:/Oracle/Middleware/user_projects/applications/mydomain/modules/medrec/assembly/target/Plan.xml')
   ```

   The WLST `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 Section A.1, "WLSTPlan Object".

4. Identify the configuration options that you want to update and their corresponding XPath values. You can determine the XPath value for configuration options by looking at the `<xpath>` element for the options in the Plan.xml file.

5. Determine if variable definitions and variable assignments are currently defined in your deployment plan for the configuration options identified in the previous step. To do so, enter one of the following commands:

   a. To display variables:
plan.showVariables()
Name                                           Value
-----                                           -----  
SessionDescriptor_cookieMaxAgeSecs_12910569321171 -1  
SessionDescriptor_invalidationIntervalSecs_12910568567990 75  
SessionDescriptor_maxInMemorySessions_12910569321170 -1  
SessionDescriptor_timeoutSecs_12900890060180 3600

b. To display variable assignments:

plan.showVariableAssignments()

medrec.ear
    |
    META-INF/weblogic-application.xml
    |  
SessionDescriptor_timeoutSecs_12900890060180
medrec.ear
    |
    META-INF/weblogic-application.xml
    |  
SessionDescriptor_invalidationIntervalSecs_12910568567990
medrec.ear
    |
    META-INF/weblogic-application.xml
    |  
SessionDescriptor_maxInMemorySessions_12910569321170
medrec.ear
    |
    META-INF/weblogic-application.xml
    |  
SessionDescriptor_cookieMaxAgeSecs_12910569321171

6. If the variable definition and assignment are not defined, create them and set the
XPath value for the variable assignment, as follows:

a. Create the variable definition. Use the createVariable() method to specify
the variable name and value. For example:

v=plan.createVariable('new_var', '3')

b. Create the variable assignment. Use the createVariableAssignment() method to specify the name of the variable, the application to which it applies,
and the corresponding deployment descriptor. For example:

va=plan.createVariableAssignment('new_var', 'medrec.ear', 'META-INF/weblogic-application.xml')
Creating VariableAssignment for ModuleOverride medrec.ear
ModuleDescriptor with URI META-INF/weblogic-application.xml.
Created VariableAssignment with name new_var successfully.

c. Set the XPath value for the variable assignment. For example:

va.setXpath('weblogic-application/session-descriptor/new_var')

7. Save the deployment plan. For example:

plan.save()
You can use WLST to retrieve information that WebLogic Server instances produce to describe their run-time state. The following sections using WLST to get run-time information:

- Section 8.1, "Accessing Runtime Information: Main Steps"
- Section 8.2, "Configuring Logging"
- Section 8.3, "Working with the WebLogic Diagnostics Framework"

### 8.1 Accessing Runtime Information: Main Steps

The Administration Server hosts the domain run-time hierarchy which provides access to any MBean on any server in the WebLogic domain. If the Administration Server is not running for a WebLogic domain, WLST can connect to individual Managed Servers to retrieve run-time data.

Accessing the run-time information for a WebLogic domain includes the following main steps:

1. Invoke WLST and connect to a running Administration Server instance. See Section 2.4.1, "Invoking WLST".

2. Navigate to the domain run-time MBean hierarchy by entering the `domainRuntime` command.

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

   The `domainRuntime` command places WLST at the root of the domain-wide run-time 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 run-time MBeans.

   ```
   wls:/mydomain/domainRuntime/ServerRuntimes/myserver>cd('JVMRuntime/myserver')>
   wls:/mydomain/domainRuntime/ServerRuntimes/myserver/JVMRuntime/myserver>ls()
   ```

   
   ```
   -r--    HeapFreeCurrent                191881368
   -r--    HeapFreePercent                87
   -r--    HeapSizeCurrent                259588096
   -r--    HeapSizeMax                    518979584
   -r--    JavaVMVendor                   Sun Microsystems Inc.
   -r--    JavaVendor                     Sun Microsystems Inc.
   ```
The following sections provide example scripts for retrieving run-time information about WebLogic Server server instances and WebLogic domain resources.

### 8.1.1 Script for Monitoring Server State

The WLST online script in Example 8–1 navigates the domain run-time hierarchy and checks the status of a Managed Server every 5 seconds. It restarts the server if the server state changes from RUNNING to any other status. It assumes that WLST is connected to the WebLogic domain’s Administration Server.

**Example 8–1 Monitoring Server State**

```python
# 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 slrBean
checkHealth("myserver")
```

### 8.1.2 Script for Monitoring the JVM

The WLST online script in Example 8–2 monitors the HJVMHeapSize for all running servers in a WebLogic domain; it checks the heap size every 3 minutes and prints a warning if the heap size is greater than a specified threshold. It assumes that the URL for the WebLogic domain’s Administration Server is `t3://localhost:7001`.

For information on how to run this script, see Section 2.4.1, "Invoking WLST".

**Example 8–2 Monitoring the JVM Heap Size**

```python
waitTime=180000
THRESHOLD=300000000
uname = "weblogic"
pwd = "welcome1"
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())
                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
            except WLSTException,e:
                # this typically means the server is not active, just ignore
                # pass
                print "Ignoring exception " + e.getMessage()
            java.lang.Thread.sleep(waitTime)

def getRunningServerNames():
    # only returns the currently running servers in the domain
    return domainRuntimeService.getServerRuntimes()

if __name__== "main":
    monitorJVMHeapSize()

8.2 Configuring Logging

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

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

The WLST online script in Example 8–3 sets attributes of LogMBean (which extends LogFileMBean). For information on how to run this script, see Section 2.4.1, "Invoking WLST".

Example 8–3  Configuring Logging

# Connect to the server
connect('weblogic','welcome1','t3://localhost:7001')
edit()
startEdit()

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

# change LogMBean attributes
set('FileCount', 5)
set('FileMinSize', 400)

# list the current directory to confirm the new attribute values
ls ()

# save and activate the changes
save()
activate()

# all done...
exit()

8.3 Working with the WebLogic Diagnostics Framework

The WebLogic Diagnostic Framework (WLDF) is a monitoring and diagnostic framework that can collect diagnostic data that servers and applications generate. You configure WLDF to collect the data and store it in various sources, including log records, data events, and harvested metrics. For more information, see Configuring and Using the Diagnostics Framework for Oracle WebLogic Server.

For example scripts that demonstrate using WLST to configure the WebLogic Diagnostic Framework, see “WebLogic Scripting Tool Examples” in Configuring and Using the Diagnostics Framework for Oracle WebLogic Server.

To view the collected diagnostics information using WLST, use one of the following commands to export the data from the WLDF repositories:

- From WLST offline, use the `exportDiagnosticData` command (see “exportDiagnosticData” in WebLogic Scripting Tool Command Reference).
- From WLST online, use the `exportDiagnosticDataFromServer` command (see “exportDiagnosticDataFromServer” in WebLogic Scripting Tool Command Reference).
The following sections describe the WLST deployment objects:

- Section A.1, "WLSTPlan Object"
- Section A.2, "WLSTProgress Object"

**A.1 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" in WebLogic Scripting Tool Command Reference.

The following table describes the WLSTPlan object methods that you can use to operate on the deployment plan.

<table>
<thead>
<tr>
<th>To operate on the...</th>
<th>Use this method...</th>
<th>To...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deployment Plan</td>
<td>DeploymentPlanBean getDeploymentPlan()</td>
<td>Return the DeploymentPlanBean for the current application.</td>
</tr>
<tr>
<td>Deployment Plan</td>
<td>void save() throws FileNotFoundException, ConfigurationException</td>
<td>Saves the deployment plan to a file from which it was read.</td>
</tr>
<tr>
<td>Module Descriptors</td>
<td>ModuleDescriptorBean createModuleDescriptor(String uri, String moduleOverrideName)</td>
<td>Create a ModuleDescriptorBean with the specified uri for the ModuleOverrideBean moduleOverrideName</td>
</tr>
<tr>
<td>Module Overrides</td>
<td>ModuleOverrideBean[] getModuleOverride(String name)</td>
<td>Return the ModuleOverrideBean name.</td>
</tr>
<tr>
<td>Module Overrides</td>
<td>ModuleOverrideBean[] getModuleOverrides()</td>
<td>Return all ModuleOverrideBean objects that are available in the deployment plan.</td>
</tr>
<tr>
<td>Module Overrides</td>
<td>void showModuleOverrides()</td>
<td>Prints all of the ModuleOverrideBean objects that are available in the deployment plan as name/type pairs.</td>
</tr>
<tr>
<td>Variables</td>
<td>VariableBean createVariable(String name, String value)</td>
<td>Create a VariableBean name with this specified value that can override the value in the deployment plan.</td>
</tr>
<tr>
<td>Variables</td>
<td>void destroyVariable(String name)</td>
<td>Destroy the VariableBean name.</td>
</tr>
<tr>
<td>Variables</td>
<td>VariableBean getVariable(String name)</td>
<td>Return the VariableBean name.</td>
</tr>
</tbody>
</table>
The WLSTProgress object enables you to check the status of an executed deployment command. The WLSTProgress object is returned by the following commands (refer to the associated command section in WebLogic Scripting Tool Command Reference for more information):

- deploy
- distributeApplication
- redeploy
- startApplication
- stopApplication
- updateApplication

The following table describes the WLSTProgress object methods that you can use to check the status of the current deployment action.

### Table A–2 WLSTProgress Object Methods

<table>
<thead>
<tr>
<th>Use this method...</th>
<th>To...</th>
</tr>
</thead>
<tbody>
<tr>
<td>String getCommandType()</td>
<td>Return the deployment CommandType of this event. This command returns one of the following values: distribute, redeploy, start, stop, or undeploy.</td>
</tr>
<tr>
<td>String getMessage()</td>
<td>Return information about the status of this event.</td>
</tr>
</tbody>
</table>
### Table A–2  (Cont.) WLSTProgress Object Methods

<table>
<thead>
<tr>
<th>Use this method...</th>
<th>To...</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProgressObject getProgressObject()</td>
<td>Return the ProgressObject that is associated with the current deployment action.</td>
</tr>
<tr>
<td>String getState()</td>
<td>Retrieve the state of the current deployment action. <strong>CommandType</strong> of this event. This command returns one of the following values: running, completed, failed, or released (indicating that the object has been released into production).</td>
</tr>
<tr>
<td>boolean isCompleted()</td>
<td>Determine if the current deployment action has been completed.</td>
</tr>
<tr>
<td>boolean isFailed()</td>
<td>Determine if the current deployment action has failed.</td>
</tr>
<tr>
<td>boolean isRunning()</td>
<td>Determine if the current deployment action is running.</td>
</tr>
<tr>
<td>void printStatus()</td>
<td>Print the current status of the deployment action, including the command type, the state, additional messages, and so on.</td>
</tr>
</tbody>
</table>
This appendix lists various frequently asked questions relating to WLST.

B.1 General WLST

On which versions of WebLogic Server is WLST supported?

WLST online is supported on WebLogic Server 11g Release 1 (10.3.1, 10.3.2, 10.3.3, and 10.3.4), 10.3, 10.0, 9.x, 8.1, and 7.0. WLST offline is supported on WebLogic Server 11g Release 1 (10.3.1, 10.3.2, 10.3.3, and 10.3.4), 10.3, 10.0, 9.x and 8.1 SP5.

What is the relationship between WLST and the existing WebLogic Server command-line utilities, such as wlconfig and weblogic.Deployer?

WLST functionality includes the capabilities of the following WebLogic Server command-line utilities:

- `wlconfig` Ant task tool for making WebLogic Server configuration changes (see "Using Ant Tasks to Configure and Use a WebLogic Server Domain" in Developing Applications for Oracle WebLogic Server)
- `weblogic.Deployer` utility for deploying applications. (see "Deployment Tools" in Deploying Applications to Oracle WebLogic Server)

When would I choose to use WLST over the other command-line utilities or the Administration Console?

You can create, configure, and manage WebLogic 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.

What is the distinction between WLST online and offline?

You can use WLST **online** (connected to a running Administration Server or Managed Server instance) and **offline** (not connected to a running server).

WLST online interacts with an active WebLogic domain and provides simplified access to Managed Beans (MBeans), WebLogic Server Java objects that you can also manage through JMX. Online, WLST provides access to information that is persisted as part of the internal representation of the configuration.

WLST offline enables you to create a new WebLogic domain or update an existing WebLogic domain without connecting to a running WebLogic Server—supporting the same functionality as the Configuration Wizard. Offline, WLST only provides access to information that is persisted in the `config` directory.
B.2 Jython Support

What version of Jython is used by WLST?

The WLST scripting environment is based on the Java scripting interpreter, Jython 2.2.1.

Can I run regular Jython scripts from within WLST?

Yes. WebLogic Server developers and administrators can extend the WebLogic scripting language to suit their environmental needs by following the Jython language syntax. For more information, see http://www.jython.org.

B.3 Using WLST

If I have SSL or the administration port enabled for my server, how do I connect using WLST?

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
```

In the event of an error, can I control whether WLST continues or exits?

Yes, using the `exitonerror` variable. Set this variable to `true` to specify that execution should exit when WLST encounters an error, or `false` to continue execution. This variable defaults to `true`. For more information, see "WLST Variable Reference" in WebLogic Scripting Tool Command Reference.

Why do I have to specify (and) after each command, and enclose arguments in single- or double-quotes?

This is the proper Jython syntax. For more information, see http://www.jython.org.

Can I start a server, deploy applications, and then shut down the server using WLST?

Yes, see documentation for the following groups of WLST commands:

- "Life Cycle Commands" in WebLogic Scripting Tool Command Reference
- "Deployment Commands" in WebLogic Scripting Tool Command Reference

Can WLST connect to a Managed Server?

Yes. You can connect to a Managed Server using the `connect` command. While connected to a Managed Server, you can view run-time data for the server and manage the security data that is in your Authentication provider's data store (for example, you can add and remove users). You cannot modify the WebLogic domain's configuration. For more information, see "connect" in WebLogic Scripting Tool Command Reference.

Can WLST use variables that I define in a properties file?

Yes. You can use the `loadProperties` command to load your variables and values from a properties file. When you use the variables in your script, during execution, the
variables are replaced with the actual values from the properties file. See "loadProperties" in WebLogic Scripting Tool Command Reference.

**Does the configToScript command convert security MBeans in config.xml?**

Yes, the security MBeans are converted. However, the information within the Embedded LDAP is not converted.

**How can I access custom MBeans that are registered in the WebLogic MBeanServer?**

To navigate to the custom MBean hierarchy on the Runtime MBean Server, use the custom command. To navigate to the custom MBean hierarchy on the Domain Runtime MBean Server, use the domainCustom command. For more information, see "Tree Commands" in WebLogic Scripting Tool Command Reference.

**Why am I not seeing all the MBeans that are registered in the MBeanServer?**

There are internal and undocumented MBeans that are not shown by WLST.

**Why does WLST offline not display the same MBeans as WLST online?**

As a performance optimization, WebLogic Server does not store most of its default values in the WebLogic domain’s configuration files. In some cases, this optimization prevents entire management objects from being displayed by WLST offline (because WebLogic Server has never written the corresponding XML elements to the WebLogic domain’s configuration files). For example, if you never modify the default logging severity level for a WebLogic domain while the domain is active, WLST offline will not display the domain’s Log management object.

If you want to change the default value of attributes whose management object is not displayed by WLST offline, you must first use the create command to create the management object. Then you can cd to the management object and change the attribute value. See “create” in WebLogic Scripting Tool Command Reference.

**When browsing custom MBeans, why do I get the following error message: No stub Available?**

When browsing the custom MBeans, the cmo variable is not available.

**Can I connect to a WebLogic Server instance via HTTP?**

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

**Can I invoke WLST via Ant?**

Yes, you can initiate a new weblogic.WLST process inside an Ant script and pass your script file as an argument.

**Can WLST scripts execute on the server side?**

Yes. You can create an instance of the WLST interpreter in your Java code and use it to run WLST commands and scripts. You can then call the WLST scripts as a startup class or as part of ejbCreate so that they execute on the server side. For more information, see Section 2.2.3, "Embedded Mode”.

**Can I customize WLST?**

Yes. You can update the WLST home directory to define custom WLST commands, WLST commands within a library, and WLST commands as a Jython module. For more information, see Section 2.9, "Customizing WLST”.

**How do I execute custom WLST commands?**
You execute custom WLST commands in the same way as WebLogic Server WLST commands. Once you define custom commands in a .py file, they are available for use from the WLST command line and within scripts.

Similarly, if you have installed Fusion Middleware (FMW) components that include custom WLST commands, the commands are available for use from the WLST command line or within scripts. For information on how to use the FMW Custom WLST commands, see "Using Custom WLST Commands" in the Oracle Fusion Middleware Administrator’s Guide.

The FMW custom commands are WLST versions of many (but not all) of the existing Oracle ASCTL commands. For a complete list and description of the available FMW commands, see WebLogic Scripting Tool Command Reference. You can display help for these commands by entering the help(), help('commandGroup'), and help('commandName') commands on the WLST command line.