Oracle® Fusion Middleware

Understanding the WebLogic Scripting Tool

14*c* (14.1.1.0.0)
F21132-06
October 2023

Oracle Fusion Middleware Understanding the WebLogic Scripting Tool, 14*c* (14.1.1.0.0)

F21132-06

Contents

Preface

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

Audience

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

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Related Documentation

For information about the WLST commands and their syntax, see:

- WebLogic Server WLST Online and Offline Command Reference in the *WLST Command Reference for WebLogic Server* for the WLST commands that are available for WebLogic Server.
- Introduction and Roadmap to the Infrastructure WLST Commands in the *WLST Command Reference for Infrastructure Components* for the WLST commands that are available for Oracle Fusion Middleware infrastructure components, including Java

Required Files (JRF), Web services, Metadata services (MDS), Application Development Framework (ADF), Dynamic Monitoring Service (DMS), Logging, Diagnostic Framework, and User Messaging Service (UMS).

- Introduction and Roadmap in the *WLST Command Reference for Infrastructure Security* for the WLST commands that are available for Oracle Fusion Middleware Infrastructure Security components, including Oracle Platform Security Services, Auditing, SSL, wallets, and OPSS Keystore Service.
- Introduction and Roadmap in the *WebCenter WLST Command Reference* for the WLST commands that are available for WebCenter components, including WebCenter Portal, WebCenter Content, and WebCenter Capture.
- Introduction and Roadmap in the *WLST Command Reference for SOA Suite* for the WLST commands that are available for SOA, Business Process Management (BPM), Enterprise Scheduler (ESS), and Managed File Transfer (MFT).
- Overview of the WebLogic Scripting Tool for Oracle Traffic Director in the *WebLogic Scripting Tool Command Reference for Oracle Traffic Director* for the WLST commands that are available for Oracle Traffic Director.
- Oracle HTTP Server WLST Custom Commands in the *Administering Oracle HTTP Server* for the WLST commands that are available for Oracle HTTP Server.

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 Online Help* describes a Web-based graphical user interface for managing and monitoring WebLogic domains.
- *Creating WebLogic 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 Using JMX for Oracle WebLogic Server* describes using Java Management Extensions (JMX) APIs to monitor and modify WebLogic Server resources.
- *Monitoring Oracle WebLogic Server with SNMP* describes using Simple Network Management Protocol (SNMP) to monitor WebLogic domains.
- *Administering Server Environments for Oracle WebLogic Server* describes how you design, configure, and manage WebLogic Server environments. It is a resource for system administrators and operators responsible for implementing a WebLogic Server installation.
- *Administering Node Manager for Oracle WebLogic Server* describes how to configure and use Node Manager to control and manage servers within a WebLogic Server environment.

Conventions

The following text conventions are used in this document:

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

1

Introduction and Roadmap

Before you get started, you should understand the the contents and organization of this guide.

Guide to This Document

You should understand the organization of this document.

It is organized as follows:

- This chapter, Introduction and Roadmap introduces the organization of this guide and lists related documentation.
- Using the WebLogic Scripting Tool describes how the scripting tool works, its modes of operation, and the basic steps for invoking it.
- 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.
- Managing the Server Life Cycle describes using WLST to start and stop WebLogic Server instances and to monitor and manage the server life cycle.
- Navigating MBeans (WLST Online) describes how to retrieve WebLogic domain configuration and run-time information, and edit configuration or custom MBeans.
- Configuring Existing WebLogic Domains describes using scripts to automate the creation and management of WebLogic domains, servers, and resources.
- Updating the Deployment Plan describes using WLST to update an application's deployment plan.
- Getting Runtime Information describes using WLST to retrieve information about the run-time state of WebLogic Server instances.
- WLST Deployment Objects describes WLST deployment objects that you can use to update a deployment plan or access information about the current deployment activity.
- FAQs: WLST provides a list of common questions and answers.
- WLST Deprecated Features lists the deprecated features for WLST.

WLST Sample Scripts

Oracle Fusion Middleware provides both WLST online and offline sample scripts.

The following sections describe the sample scripts that you can run or use as templates for creating additional scripts:

For information about running scripts, see Running Scripts.

> **Note:**
>
> The sample scripts are not installed by default. To install the server samples, you must select the **Complete With Examples** option when installing WebLogic Server.

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: *ORACLE_HOME*\\wlserver\samples\server\examples\src\examples\wlst\online.

Table 1-1 summarizes WLST online sample scripts.

Table 1-1 WLST Online Sample Scripts

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

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:
ORACLE_HOME\wlserver\common\templates\scripts\wlst.

Table 1-2 summarizes WLST offline sample scripts.

Table 1-2 WLST Offline Sample Script

WLST Sample Script	Description
`basicWLSdomain.py`	Creates a simple WebLogic domain demonstrating how to open a domain template, create and edit configuration objects, and write the domain configuration information to the specified directory. The sample consists of a single server, representing a typical development environment. This type of configuration is not recommended for production environments. The script uses the Basic WebLogic Server Domain template.
`basicWLSSdomain.py`	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.
`clusterMedRecDomain.py`	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.
`distributedQueue.py`	Demonstrates two methods for creating distributed queues. The script uses the Basic WebLogic Server Domain template and extends it using the Avitek Medical Records Sample extension template.
`geo1Domain.py`	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.
`geo2Domain.py`	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.
`replicatedDomain.py`	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.
`sampleMedRecDomain.py`	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.

2

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).

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).

The following sections describe how to use WLST online or offline:

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 a management interface for an underlying resource. For information on WebLogic Server MBeans, see Understanding WebLogic Server MBeans in *Developing Custom Management Utilities Using JMX for Oracle WebLogic Server*.

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 Browsing Information About the Configuration Hierarchy (Offline).

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 *WLST Command Reference for WebLogic Server*.

Interactive Mode, Script Mode, and Embedded Mode

WLST can be invoked in a variety of ways.

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

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. See startRecording and stopRecording in *WLST Command Reference for WebLogic Server*.

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 WLST Sample Scripts.

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. Prior to running the program with embedded WLST, you must invoke the following command to set the appropriate environment 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 or bash shell.

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

```
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('adminusername','adminpassword')");
    interpreter.exec(buffer.toString());
  }

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

```
    interpreter.exec(buf.toString());
  }

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

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

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

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:

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 *Administering Server Environments for Oracle WebLogic Server*.

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.

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 *MBean Reference for Oracle WebLogic Server*.)

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('adminusername', 'adminpassword', 'localhost:7001'
  ```

  See connect in *WLST Command Reference for WebLogic Server*.

- Enter the credentials on the command line, and 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('adminusername', 'adminpassword', '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')
  ```

  See connect and storeUserConfig in *WLST Command Reference for WebLogic Server*.

- 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()
```

See connect in *WLST Command Reference for WebLogic Server*.

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', '<password>')
  ```

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

  See set in *WLST Command Reference for WebLogic Server*.

- 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              ********
  ...
  ```

  While

  ```
  wls:/offline/wl_server/Server/
  examplesServer>get('DefaultIIOPPasswordEncrypted')
  ```

  returns

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

  See ls and get in *WLST Command Reference for WebLogic Server*.

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('DefaultIIOPPassword', '<password>')
    ```

  - 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 *WLST Command Reference for WebLogic Server*). For example:

    ```
    set('DefaultIIOPPasswordEncrypted', encrypt('<password>'))
    ```

  - **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('DefaultIIOPPasswordEncrypted')
  ```

  returns

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

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 files. 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. See Managing Security Data (WLST Online).

Main Steps for Using WLST in Interactive or Script Mode

When you use WLST interactive or script mode, you should understand how to invoke it, exit it, and its syntax.

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

Invoking WLST

You can invoke WLST in the following ways:

- Execute the appropriate shell script for your environment and enter WLST commands from the WLST shell. See Invoking WLST Using Provided Shell Scripts.
- Execute the `java weblogic.WLST` command. See Invoking WLST Using the java Command.
- Run a prepared WLST script when invoking the WLST shell script. See Running Scripts.
- 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 Running WLST from Ant.

Invoking WLST Using Provided Shell Scripts

To invoke WLST using a shell script, execute the command that is appropriate for your environment. This is the recommended way to invoke WLST. Environment variables are automatically set when you invoke WLST this way. You can then enter WLST commands from within the WLST shell.

> **Note:**
>
> The following commands apply only to standalone WebLogic Server installations (those that do not include other Fusion Middleware components). Many Fusion Middleware components supply custom WLST commands. To use them, you must invoke WLST from the appropriate directory. See Using Custom WLST Commands in *Administering Oracle Fusion Middleware*.

UNIX

```
cd ORACLE_HOME/oracle_common/common/bin
./wlst.sh
```

ORACLE_HOME is the Oracle Home directory you specified at installation.

To set WLST properties in a UNIX environment prior to running WLST:

```
export WLST_PROPERTIES="-Dweblogic.security.SSL.ignoreHostnameVerification=true,
-Dweblogic.security.TrustKeyStore=DemoTrust"
./wlst.sh
```

> **Note:**
>
> To avoid log file name collisions when multiple WLST offline processes are running, set the following property in the environment prior to invoking WLST:
>
> ```
> export WLST_PROPERTIES="-Dwlst.offline.log=./logs/debug.log"
> ```

Windows

```
cd ORACLE_HOME\oracle_common\common\bin
wlst.cmd
```

ORACLE_HOME is the Oracle Home directory you specified at installation.

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 Installing the Oracle WebLogic Server and Coherence Software in *Installing and Configuring Oracle WebLogic Server and Coherence*.
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 full path to the WebLogic Server home directory.

   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 or bash 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. Table 2-3 lists additional optional startup arguments for WLST.

```
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 ]
[ -skipWLSModuleScanning ]
[ [-i] filePath.py ]
```

Table 2-1 describes the WLST command options.

Table 2-1 Command Options for WLST

Option	Description
`-loadProperties` *`propertyFilename`*	Use this option to load properties into the WLST session, where *`propertyFilename`* is the name of a file that contains `name=value` pairs. You cannot use this option when you are importing WLST as a Jython module (see Importing WLST as a Jython Module). Instead of using this command-line option, you can use the `loadProperties` WLST command. See loadProperties in *WLST Command Reference for WebLogic Server*.
`-skipWLSModuleScanning`	Use this option to reduce startup time by skipping package scanning and caching for WebLogic Server modules.
`-[-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: `execfile('`*`filePath`*`.py').`

Table 2-2 describes the SSL arguments.

Table 2-2 SSL Arguments

Argument	Definition
`-Dweblogic.security.SSL.ignoreHostnameVerification=true -Dweblogic.security.TrustKeyStore=DemoTrust`	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. `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`).
`-Dweblogic.security.JavaStandardTrustKeyStorePassPhrase=`*`password`*	Password that was used to secure the Java Standard Trust keystore. If the Java Standard Trust keystore is protected by a password, and if you want to trust its CA certificates, you must use this argument. By default, the Java Standard Trust keystore is not protected by a password.
`-Dweblogic.security.CustomTrustKeyStoreFileName=`*`filename`*`-Dweblogic.security.TrustKeystoreType=jks`	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.
`-Dweblogic.security.CustomTrustKeyStorePassPhrase=`*`password`*	Password that was used to secure the custom keystore. You must use this argument only if the custom keystore is protected by a password.
`-Dweblogic.security.SSL.hostnameVerifier=`*`classname`*	Name of a custom Host Name Verifier class. The class must implement the `weblogic.security.SSL.HostnameVerifier` interface.

Table 2-3 describes additional optional startup arguments for WLST.

Table 2-3 Additional WLST Startup Arguments

Argument	Description
`-Dwlst.debug.init=`*`value`*	If `true`, debug mode is turned on. If `false`, debug mode is turned off. The default is `false`.
`-Dweblogic.wlstHome=`*`path1: path2: ...: pathn`* `-Dweblogic.wlstHome=`*`path1; path2; ...; pathn`*	A comma-separated list of directory paths from which to load `.py` files at startup. Note that the path separator is '`:`' on UNIX operating systems and '`;`' on Windows.

Table 2-3 (Cont.) Additional WLST Startup Arguments

Argument	Description
`-Dwlst.offline.log=path`	Sets the path and file name for WLST offline logging, for example, `-Dwlst.offline.log=./logs/debug.log`.
`-Dwlst.offline.log.priority=debug`	Sets the log priority level to debug for WLST offline logging.

Example 2-2 Examples of Invoking WLST Using the Java Command

To use WLST in script mode:

```
java weblogic.WLST c:/Oracle/Middleware/wlserver/common/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/common/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('adminusername','adminpassword','localhost:7001')
```

Running Scripts

You can run WLST scripts in the following ways:

- Include the script in the `wlst.cmd` or `wlst.sh` command:

  ```
  wlst.sh  /oracle/middleware/wlserver/common/templates/scripts/wlst/
  distributedQueues.py
  ```

- 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/common/templates/scripts/
  wlst/distributedQueues.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/common/templates/scripts/wlst
  java weblogic.WLST distributeQueues.py
  ```

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

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

> **Note:**
>
> If you use backslashes (\) in the path name, be aware that WLST interprets them as a special character, which you must escape. See Syntax for WLST Commands.

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**).

Exiting WLST

To exit WLST, enter the `exit()` command:

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

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', 'Expanded')
  ```

  or

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

  > **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', 'Expanded')
  > ```
  >
  > is as valid as
  >
  > ```
  > readTemplate('c:\\userdomains\\templates\\template.jar', 'Expanded')
  > ```

- 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)')
```

Considerations When Invoking Multiple WLST Instances

At WLST startup, Jython stores information in a temporary directory based on the username of the person who started WLST. If the same user invokes two different WLST instances which will run at the same time, conflicts may occur when saving this information in the temporary directory.

If you plan to invoke multiple WLST instances with the same username, Oracle recommends that you define the `java.io.tmpdir` system property to point to a temporary directory that will not be shared by other WLST instances that are running at the same time. For example, include the following parameter in the Java command you use to start WLST:

```
-Djava.io.tmpdir=C:\mytempdir
```

Redirecting Error and Debug Output to a File

You can redirect error and debug information to a file.

To redirect WLST information, error, and debug messages from standard output 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:

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

See redirect and stopRedirect in *WLST Command Reference for WebLogic Server*.

Getting Help for WLST

WLST provides command line help for each command.

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

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

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

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

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

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

The command returns the following:

```
Description:
Disconnect from a WebLogic Server instance.
Syntax:
disconnect()
Example:
wls:/mydomain/serverConfig> disconnect()
```

Running WLST from Ant

WebLogic Server provides a custom Ant task, `wlst`, which 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, where *`WL_HOME`* is the directory in which you installed WebLogic Server.

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

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 or bash shell.

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" />
```

WLST Task Parameters

Table 2-4 lists the `wlst` task parameters that you specify as attributes of the `<wlst>` element.

Table 2-4 wlst Parameters

Attribute	Description	Required
properties="*`propsFile`*"	Name and location of a properties file that contains name-value pairs that you can reference in your WLST script.	No
fileName="*`fileName`*"	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.	Yes, if no nested `<script>` is used.

Table 2-4 (Cont.) wlst Parameters

Attribute	Description	Required
arguments="*arglist*"	List of arguments to pass to the script. These arguments are accessible using the `sys.argv` variable.	No
failOnError="*value*"	Boolean value specifying whether the Ant build will fail if this task fails.	No; default is `true`.
executeScriptBeforeFile="*value*"	Boolean value specifying whether this task invokes the script in the nested `<script>` element before the script file specified by the `fileName` attribute. This attribute defaults to `true`, specifying that the embedded script is invoked first.	No; default is `true`.
debug="*value*"	Boolean value specifying whether debug statements should be output when this task is executed.	No; default is `false`.
replaceProperties="*value*"	Boolean value that specifies whether ant property expansion will work in the specified WLST script.	No; default is `true`.

WLST Task Parameters Specified as Nested Elements

Table 2-5 describes the `wlst` task parameters that you specify as nested elements of the `<wlst>` element.

Table 2-5 Parameters Specified as Nested Elements

Attribute	Description	Required
script	The script to be executed.	Required, if you do not use the fileName attribute to name a script file.
classpath	The classes to add to the classpath. This element is the standard Ant `classpath` element. You can specify a reference to a `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 *Apache Ant 1.7.1 Manual* at `http://ant.apache.org/manual/using.html#path`	No. Use if your script requires classes that are not already on the classpath.

WLST Ant Task Examples

The following sections show examples for the `createServer`, `loop`, and `error` targets.

CreateServer Target Example

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.

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

Loop Target Example

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.

```
<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="${my.classpath.dir}"/>
    </classpath>
    <script replaceProperties="true">
       print 'In the target loop'
       connect('${admin.user}','${admin.password}','t3://localhost:7001')
       svrs = cmo.getServers()
       print 'Servers in the domain are'
```

```
        for x in svrs: print x.getName()
    </script>
  </wlst>
</target>
```

Error Target Example

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>
```

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.

   ```
   cd ORACLE_HOME/oracle_common/common/bin
   ./wlst.sh
   wls:/offline>
   ```

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

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

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

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

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

   ```
   >>>
   ```

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

   ```
   >>>import wl
   ```

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

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

> **Note:**
>
> When using WLST as a Jython module, in all WLST commands that have a block argument, block is always set to true, specifying that WLST will block user interaction until the command completes. See WLST Command and Variable Reference in *WLST Command Reference for WebLogic Server*.
>
> 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:
> - `true=1`
>   `false=0`
> - Change `true` and `false` values to `wl.true` or `wl.false`, where `wl` is the name from the import.

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`*`>`

Table 2-6 describes ways to customize WLST.

Table 2-6 Customizing WLST

To define custom...	Do the following...	For a sample script, see...
WLST commands	Create a Python script defining the new commands and copy that file to *`WL_HOME`*`/common/wlst`.	Sample Scripts For Defining new WLST Commands

Table 2-6 (Cont.) Customizing WLST

To define custom...	Do the following...	For a sample script, see...
WLST commands within a library	Create a Python script defining the new commands and copy that file to *WL_HOME*/common/wlst/lib. The scripts located within this directory are imported as Jython libraries.	Sample Scripts For Defining new WLST Commands
WLST commands as a Jython module	Create a Python script defining the new commands and copy that file to *WL_HOME*/common/wlst/modules. This script can be imported into other Jython modules, as described in Importing WLST as a Jython Module.	*WL_HOME*/common/wlst/modules/wlstModule.py A JAR file, jython-modules.jar, which contains all of the Jython modules that are available in Jython 2.7.1, is also available within this directory. To import wlstModule, see Import wlstModule for Modularity.
Integrated help for custom WLST commands	Define the help text for each command group and command in a resource bundle. 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 Adding Integrated Help for Custom Commands.	Not applicable.

Adding Integrated Help for Custom Commands

You can customize WLST to include integrated help for any custom WLST 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:

```
<commandGroup>_ShortDescription=\
     <short description of command group>
```

```
<commandGroup>_Description=\
     \n<description of command group>
```

For example, the following lines define the short description and description for the command group `navigate`:

```
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:

```
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:

```
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:

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

The following defines the help for `mycd` command:

```
mycd_ShortDescription=\
     Navigate the hierarchy of beans.

mycd_Description=\
     \nNavigate the hierarchy of configuration or runtime beans. This \
     \ncommand uses a model that is similar to navigating a file system \
     \nin a Windows or UNIX command shell. For example, to navigate back \
     \nto a parent configuration or runtime bean, enter mycd(".."). The \
     \ncharacter string .. (dot-dot) refers to the directory immediately \
     \nabove the current directory. To get back to the root configuration \
     \nbean after navigating to a configuration or runtime bean that is \
     \ndeep in the hierarchy, enter mycd("/"). \

mycd_Example=\
     wls:/mydomain/serverConfig> mycd('Servers')\n\
     wls:/mydomain/serverConfig/Servers> mycd('myserver)\n\
     wls:/mydomain/serverConfig/Servers/myserver>mycd('../..')\n\
     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:

```
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:

```
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:

wls:/mydomain/serverConfig> mycd('Servers')\n\
wls:/mydomain/serverConfig/Servers> mycd('myserver)\n\
wls:/mydomain/serverConfig/Servers/myserver>mycd('../..')\n\
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 *WLST Command Reference for WebLogic Server*.

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

Sample Scripts For Defining new WLST Commands

The following sample script demonstrates how to define a new WLST command.

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

Within this script, the `wlstHomeSample()` command is defined, which prints a text string:

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

When defining new WLST commands in this way, store the `.py` file in the *WL_HOME*/common/wlst directory.

The following sample script demonstrate usage of the WLST `lib` directory, where layered products and ISVs can add commands to WLST in their namespace. The script has the same structure as the previous example. It differs only in that it is stored in the *WL_HOME*/common/wlst/lib directory instead of the *WL_HOME*/common/wlst directory.

```
def wlstExampleCmd():
     print 'Example command'
```

Within this script, the `wlstExampleCmd()` command is defined, which prints a text string:

```
wls:/offline> wlstLibSample.wlstExampleCmd()
Example command
```

Note that you must create the WLST `lib` directory if it does not already exist.

3

Creating WebLogic Domains Using WLST Offline

You can create and configure WebLogic domains using WebLogic Scripting Tool (WLST) offline. 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.
For information about sample scripts that you can use to create WebLogic domains, see WLST Offline Sample Scripts.

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

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.

> **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`

To create and use a domain template, perform the steps described in Table 3-1.

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

To...	Use this command...	See...

Open an existing WebLogic domain or select the templates for the domain	`readDomain(`*`domainDirName`*`)`
	`selectTemplate(`*`templateName, TemplateVersion`*`)`
	`loadTemplates()`
		readDomain, selectTemplate and loadTemplates in *WLST Command Reference for WebLogic Server*

(Optional) Modify the WebLogic domain	Browsing and editing commands	Browsing Information About the Configuration Hierarchy (Offline) Editing a WebLogic Domain (Offline)

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

To...	Use this command...	See...
Set the password for the default user, if it is not already set. The default username and password must be set before you can write the domain template.	`cd('/Security/domainname/User/username')` `cmo.setPassword('password')`	WLST Offline Sample Scripts
Write the domain configuration information to a domain template.	`writeTemplate(templateName)`	writeTemplate in *WLST Command Reference for WebLogic Server*
Use the template to create a domain.	`createDomain(domainTemplate, domainDir, user, password)` **Note:** The Configuration Wizard can also use the domain template. See Introduction to WebLogic Domains in *Creating WebLogic Domains Using the Configuration Wizard*.	createDomain in *WLST Command Reference for WebLogic Server*

Example 3-1 shows the basic commands needed to create a domain template from an existing domain:

Example 3-1 Creating a Domain Template From an Existing Domain

```
# Read the domain from the domain home located at /domains/mydomain
readDomain('/domains/mydomain')

# Create the template jar mydomain.jar in /templates
writeTemplate('/templates/mydomain.jar')
```

Creating and Updating a WebLogic Domain

When creating or updating a WebLogic domain, there are three phases to the configuration session:

- Load and merge the configuration using readTemplate (for creating) or readDomain (for updating) optionally followed by one or more addTemplate() calls.
- Modify the configuration by navigating the configuration tree and updating attributes.
- Save the configuration using writeDomain (for creating), updateDomain (for updating), or writeTemplate (for creating a template), followed by closeTemplate or closeDomain as appropriate.

The following examples demonstrate the sequence of commands for creating a domain from a single template and creating a domain from multiple templates. More extensive sample WLST offline scripts are available in the following directory:

ORACLE_HOME/wlserver/common/templates/scripts/wlst

Example 3-2 shows the basic sequence of commands needed to create a domain from a single template. This example creates a basic WebLogic Server domain from the wls.jar template.

Example 3-2 Creating a Domain From a Single Template

```
# Read the template to use for creating the domain
readTemplate('oracle_home/middleware/wlserver/common/templates/wls/wls.jar')

# Set the listen address and listen port for the Administration Server
cd('Servers/AdminServer')
set('ListenAddress','')
set('ListenPort', 7001)

# Enable SSL on the Administration Server and set the SSL listen address and
# port
create('AdminServer','SSL')
cd('SSL/AdminServer')
set('Enabled', 'True')
set('ListenPort', 7002)

# Set the domain password for the WebLogic Server administration user
cd('/')
cd('Security/base_domain/User/adminusername')
cmo.setPassword('adminpassword')

# If the domain already exists, overwrite the domain
setOption('OverwriteDomain', 'true')

# write the domain and close the template
writeDomain('/domains/mydomain')
closeTemplate()

exit()
```

Example 3-3 shows how to create a new domain using a domain template and one or more extension templates. This example creates the basic WebLogic Server domain from the wls.jar template and then extends the domain by adding the WebLogic Web Services Extension template to the domain.

Example 3-3 Creating a Domain From Multiple Templates

```
# Select the WebLogic domain template, and
# then load it
selectTemplate('Basic WebLogic Server Domain')
selectTemplate('WebLogic Advanced Web Services for JAX-RPC Extension')
loadTemplates()
# Set the listen address and listen port for the Administration Server
cd('Servers/AdminServer')
set('ListenAddress','')
set('ListenPort', 7001)

# Enable SSL on the Administration Server and set the SSL listen address and
# port
create('AdminServer','SSL')
cd('SSL/AdminServer')
set('Enabled', 'True')

set('ListenPort', 7002)

# Set the domain password for the WebLogic Server administration user
cd('/')
cd('Security/base_domain/User/adminusername')
cmo.setPassword('adminpassword')
```

```
# If the domain already exists, overwrite the domain
setOption('OverwriteDomain', 'true')

# write the domain and close the templates
writeDomain('/domains/mydomain')

# Select the WebServices for JAX-RPC template, and
# then load it

selectTemplate('WebLogic Advanced Web Services for JAX-RPC Extension')
loadTemplates()
closeTemplate()

exit()
```

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/security/1.0/security.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.

> **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 *WLST Command Reference for WebLogic Server*.

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).

Table 3-2 Displaying WebLogic Domain Configuration Information (Offline)

To...	Use this command...	See this section in *WLST Command Reference for WebLogic Server*
Navigate the hierarchy of management objects	`cd(path)`	cd
List child attributes or management objects for the current management object	`ls(['a' | 'c'])`	ls
Toggle the display of the management object navigation path information at the prompt	`prompt(['off'|'on'])`	prompt
Display the current location in the configuration hierarchy	`pwd()`	pwd
Display all variables used by WLST	`dumpVariables()`	dumpVariables
Display the stack trace from the last exception that occurred while performing a WLST action	`dumpStack()`	dumpStack

Editing a WebLogic Domain (Offline)

You can edit a WebLogic domain offline.

To edit a WebLogic domain using WLST offline, you can perform any of the tasks defined in Table 3-3.

Table 3-3 Editing a WebLogic Domain

To...	Use this command...	See this section in *WLST Command Reference for WebLogic Server*
Add an application to a WebLogic domain	`selectTemplate(templateName)` `loadTemplate(templateName)`	selectTemplate and loadTemplates
Assign resources to one or more destinations (such as assigning servers to clusters)	`assign(sourceType, sourceName, destinationType, destinationName)`	assign
Unassign resources	`unassign(sourceType, sourceName, destinationType, destinationName)`	unassign
Create and delete management objects	`create(name, childMBeanType)` `delete(name, childMBeanType)`	create delete
Get and set attribute values	`get(attrName)` `set(attrName, value)`	get set
Set configuration options	`setOption(optionName, value)`	setOption
Load SQL files into a database	`loadDB(dbVersion, connectionPoolName)`	loadDB

> **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`

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 *WLST Command Reference for WebLogic Server*.

> **Note:**
>
> If you use `configToScript` for a domain that contains other Fusion Middleware components in addition to WebLogic Server, be aware that `configToScript` does not include the configuration for those components in the resulting WLST script. Only the WebLogic Server configuration is included in the script.
>
> `configToScript` will be deprecated in a future release. Oracle recommends that you use `pack` and `unpack` to recreate the domain on remote servers. See Overview of the Pack and Unpack Commands in *Creating Templates and Domains Using the Pack and Unpack Commands*.

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.

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:

```
(Windows) ORACLE_HOME\user_projects\domains\wl_server\setExamplesEnv.cmd
(UNIX) ORACLE_HOME/user_projects/domains/wl_server/setExamplesEnv.sh
```

Creating a Managed Server Domain on a Remote Machine

If your WebLogic domain contains multiple Managed Servers, and each Managed Server domain directory is located on a remote machine on which the Administration Server does not reside, you can use the WLST `writeTemplate` command in online mode. When you execute the `writeTemplate` command while connected to the Administration Server from a remote machine, it dynamically packs the domain on the Administration Server into a template JAR file and transfers the template JAR to the specified directory.

The following sample WLST script demonstrates how to use `writeTemplate` to create or update a Managed Server domain on a remote machine. Run the script on each remote machine in the domain.

```
import os

wlsHome = os.getenv('WL_HOME')
mwHome = os.path.join(wlsHome, '..')

#Substitute the administrator user name and password values below as needed
connect('adminusername','adminpassword','localhost:7001')

#The path on the local machine where the template will be created,
#it should not already exist.
templatePath = 'user_templates/myTemplate.jar'

#get the packed template from the Administration Server
writeTemplate(templatePath)

#disconnect from online WLST connection to the Administration Server
disconnect()

#select and load the template that was downloaded from the Administration
#Server.
selectCustomTemplate(templatePath)

loadTemplates()

#specify the domain directory where the domain needs to be created
domainPath = 'domains/myRemoteDomain')

#create the domain
writeDomain(domainPath)
```

> **Note:**
>
> When you use the `writeDomain()` command to copy the domain to the remote node, you must have write permissions to the `applications` directory (`c:/Oracle/Middleware/user_projects/applications/mydomain/`).

4

Managing the Server Life Cycle

You can use the WebLogic Scripting Tool (WLST) to manage and monitor the server life cycle. During its lifetime, a server can transition through a number of operational states, such as shutdown, starting, standby, admin, resuming, and running.
For more information about the server life cycle, see Understanding Server Life Cycle in *Administering Server Startup and Shutdown for Oracle WebLogic Server*.

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

Using WLST and Node Manager to Manage Servers

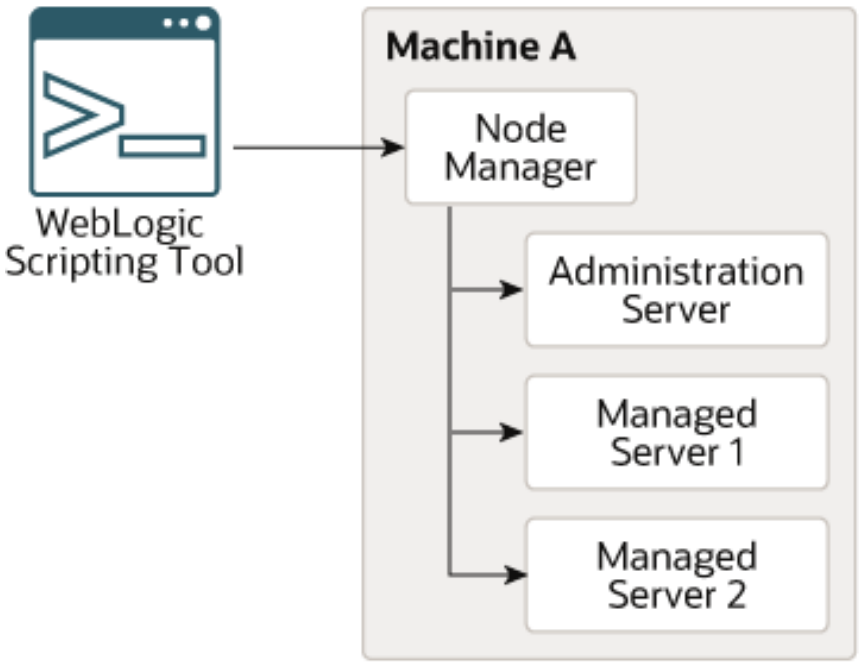
Node Manager is a utility that enables you to control the life cycles of multiple servers through a single WLST session and a single network connection. (It can also automatically restart servers after a failure.)

For more information about Node Manager, see Node Manager Overview in *Administering Node Manager 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.

Figure 4-1 Starting Servers on a Machine

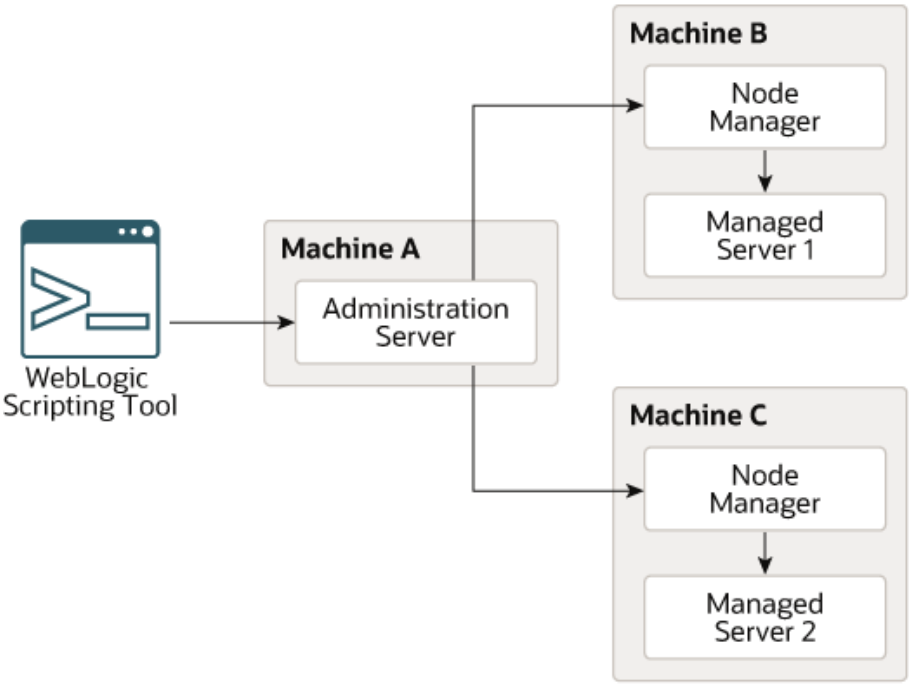


A Node Manager process may be associated with a specific WebLogic domain or it may be associated with a particular machine. If associated with a domain, you can use the Node Manager process only to control server instances in that domain. If associated with a machine, you can use the same Node Manager 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 *Administering Node Manager for Oracle WebLogic Server*. For information about Node Manager configuration, see Default Node Manager Configuration *Administering Node Manager for Oracle WebLogic Server*.

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

Figure 4-2 Starting Servers in a WebLogic Domain



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 *WLST Command Reference for WebLogic Server*.

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.

   See Configuring Java Node Manager in *Administering Node Manager for Oracle WebLogic Server*.

2. Start WLST.

3. Start Node Manager.

   If Node Manager is not already running, use the `startNodeManager` script in either *`DOMAIN_HOME`*/bin or *WL_HOME*/server/bin to start it. See Starting Java-based Node Manager Using Scripts in *Administering Node Manager for Oracle WebLogic Server*.

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('adminusername', 'adminpassword', '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 the `nmConnect` command arguments, see nmConnect in *WLST Command Reference for WebLogic Server*.

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
   wls:/nm/mydomain>
   ```

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 *WLST Command Reference for WebLogic Server*.

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.

See Configuring Java Node Manager in *Administering Node Manager for Oracle WebLogic Server*.

2. Start WLST.
3. Start Node Manager.

   If Node Manager is not already running, use the `startNodeManager` script in either *`domain_home`*`/bin` or *`WL_HOME`*`/server/bin` to start it. For more information, see Starting Java-based Node Manager Using Scripts in *Administering Node Manager for Oracle WebLogic Server*.
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 *WLST Command Reference for WebLogic Server*.
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.

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

   See start in *WLST Command Reference for WebLogic Server*.

Starting and Managing Servers Without Node Manager

You can start and manage an Administration Server 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 *Administering Server Startup and Shutdown for Oracle WebLogic Server*.

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

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.

See 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.
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 Invoking WLST.

   The WLST prompt appears.

   ```
   wls:/offline>
   ```
6. Use the WLST `startServer` command to start the Administration Server.

   ```
   startServer([adminServerName], [domainName], [url], [adminusername],
   [adminpassword],[domainDir], [block], [timeout], [serverLog],
   [systemProperties], [jvmArgs] [spaceAsJvmArgsDelimiter])
   ```

   For detailed information about `startServer` command arguments, see startServer in *WLST Command Reference for WebLogic Server*.

   For example,

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

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

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 *Administering Node Manager for Oracle WebLogic Server*. Oracle recommends that you enable and use the WebLogic domain's administration port when you connect to servers and issue administrative commands. See 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

```
# Specify the SSL arguments when starting WLST
export WLST_PROPERTIES="-Dweblogic.security.TrustKeyStore=DemoTrust,   -
Dweblogic.security.SSL.ignoreHostnameVerification=true"

./wlst.sh

# 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")

# reconnect to the server
exit()
connect("username","password","t3://localhost:7001")

# 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")
```

5

Navigating MBeans (WLST Online)

You can navigate, interrogate, and edit MBeans using WebLogic Scripting Tool (WLST) online.

Navigating and Interrogating MBeans

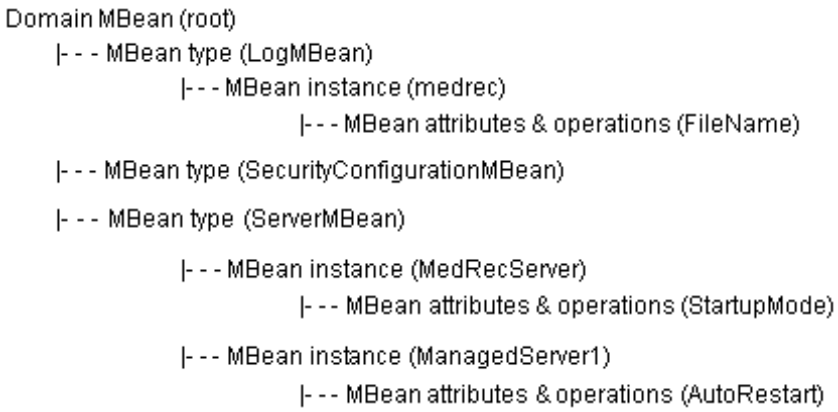
WLST online provides simplified access to MBeans. While JMX APIs require you to use JMX object names to interrogate MBeans, WLST enables you to navigate a hierarchy of MBeans in a fashion similar to navigating a hierarchy of files in a file system.

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 way that's similar to how 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 *MBean Reference for Oracle WebLogic Server*). 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



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. See Tree Commands in *WLST Command Reference for WebLogic Server*.

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

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:

```
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

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

For more information on WLST variables, see WLST Variable Reference in *WLST Command Reference for WebLogic Server*.

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`.

> **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 *MBean Reference for Oracle WebLogic Server*.

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

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

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

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

Example 5-2 Navigating and Displaying Configuration MBeans

```
wls:/offline> connect('username','password')
wls:/mydomain/serverConfig> ls()
dr--   AdminConsole
dr--   AppDeployments
dr--   BridgeDestinations
dr--   Clusters
dr--   CoherenceClusterSystemResources
dr--   CoherenceServers
dr--   CustomResources
dr--   DeploymentConfiguration
dr--   Deployments
...
-r--   AdminServerName                                myserver
-r--   AdministrationMBeanAuditingEnabled             false
-r--   AdministrationPort                             9002
-r--   AdministrationPortEnabled                      false
-r--   AdministrationProtocol                         t3s
-r--   ArchiveConfigurationCount                      5
...
wls:/mydomain/serverConfig> cd('Servers')
wls:/mydomain/serverConfig/Servers> ls()
dr--   AdminServer
dr--   managed1
dr--   myserver
wls:/mydomain/serverConfig/Servers> cd('myserver')
wls:/mydomain/serverConfig/Servers/myserver> ls()
dr--   COM
dr--   CandidateMachines
dr--   Cluster
dr--   CoherenceClusterSystemResource
dr--   DefaultFileStore
dr--   ExecutiveQueues
dr--   FederationServices
```

```
dr--   IIOP
dr--   JTAMigrateableTarget
dr--   Log
dr--   Machine
dr--   NetworkAccessPoints
...
-r--   AcceptBacklog                                50
-r--   AdminReconnectIntervalSeconds                10
-r--   AdministrationPort                           0
-r--   AdministrationProtocol                       t3s
-r--   AutoKillIfFailed                             false
-r--   AutoMigrationEnabled                         false
-r--   AutoRestart                                  true
....
wls:/mydomain/serverConfig/Servers/myserver> cd('Log/myserver')
wls:/mydomain/serverConfig/Servers/myserver/Log/myserver> ls()
dr--   DomainLogBroadcastFilter
dr--   LogFileFilter
dr--   MemoryBufferFilter
dr--   StdoutFilter

-r--   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.

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.

See ServerRuntimeMBean and DomainRuntimeMBean in the *MBean Reference for Oracle WebLogic Server*.

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

```
dr--   WorkManagerRuntimes

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

-r--   ActivationTime                              Tue Aug 31 09:27:22 EDT 2004
-r--   Clusters                                    null
-rw-   CurrentClusterDeploymentTarget             null
-rw-   CurrentClusterDeploymentTimeout           0
-rw-   Name                                        mydomain
-rw-   Parent                                      null
-r--   Type                                        DomainRuntime

-r-x   lookupServerLifecycleRuntime                javax.management.ObjectName

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

The commands in 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
dr--   JVMRuntime
dr--   LibraryRuntimes
dr--   MailSessionRuntimes
dr--   RequestClassRuntimes
dr--   ServerChannelRuntimes
dr--   ServerSecurityRuntime
dr--   ThreadPoolRuntime
```

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

Navigating Among MBean Hierarchies

You can navigate among the MBean hierarchies.

To navigate to a configuration MBean from the run-time hierarchy, enter the `serverConfig` command or, if connected to an Administration Server only, the `domainConfig` 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:

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

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

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

See Tree Commands in *WLST Command Reference for WebLogic Server*.

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 *WLST Command Reference for WebLogic Server*.

For example:

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

Finding MBeans 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. See find and getMBean in *WLST Command Reference for WebLogic Server*.

For example:

```
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
wls:/mydomain/edit !>
```

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

Alternatively, the `getPath` command returns the MBean path for a specified MBean instance or `ObjectName` for the MBean in the current MBean hierarchy. See getPath in *WLST Command Reference for WebLogic Server*.

For example:

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

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 Using 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

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 Using WLST Online to Update an Existing WebLogic Domain. Note that this applies to the `custom()` tree, but not to the `editCustom()` tree.

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

```
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()
-rw-   MyAttribute      10
wls:/mydomain/custom/mycompany/mycompany:y1=x>
```

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 Accessing Other WebLogic MBeans and Custom MBeans. You can use the `domainCustom()` command only when WLST is connected to the Administration Server.

For information on using `domainCustom()`, see domainCustom in *WLST Command Reference for WebLogic Server*.

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 Using JMX for Oracle WebLogic Server*. For information on creating and registering your own MBeans, see Instrumenting and Registering Custom MBeans in *Developing Manageable Applications Using JMX for Oracle WebLogic Server*.

Accessing Custom MBeans in the Edit MBean Server

Use the `editCustom()` command to browse, change and invoke methods or perform operations on custom MBeans that are registered in the Edit MBean Server. This is similar to

using the `custom()` command, but custom MBeans in the Edit MBeanServer also allow edit operations. You can use the `editCustom()` command only when WLST is connected to the Administration Server.

For more information about `editCustom()`, see editCustom in the *WLST Command Reference for WebLogic Server*.

Oracle system components, such as Oracle HTTP Server, may define custom MBeans that are present in the Edit MBeanServer. These MBeans are editable and are subject to the WebLogic Server change-managed process.

6
Configuring Existing WebLogic Domains

You can use WebLogic Scripting Tool (WLST) both online and offline to update an existing WebLogic domain.

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 6-1 Steps for Updating an Existing WebLogic Domain (Online)

To...	Use this command...	See this section in ***WLST Command Reference for WebLogic Server***:
Access the edit MBean hierarchy	`edit()` This command places WLST at the root of the edit MBean hierarchy, which is the editable `DomainMBean`.	edit
Obtain a lock on the current configuration To indicate that configuration changes are in process, an exclamation point (!) appears at the end of the WLST command prompt.	`startEdit([waitTimeInMillis], [timeoutInMillis], [exclusive])`	startEdit
Modify the WebLogic domain	Browsing and online editing commands	Browse Commands Editing Commands
(Optional) Validate your edits	`validate()`	validate
Save your changes	`save()`	save
Distribute your changes to the working configuration MBeans on all servers in the WebLogic domain	`activate([timeout], [block])`	activate
Release your lock on the configuration	`stopEdit([defaultAnswer])`	stopEdit

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

To...	Use this command...	See this section in *WLST Command Reference for WebLogic Server*:
(Optional) Determine if a change you made to an MBean attribute requires you to re-start servers You can use the showChanges command to determine the changes you made to the configuration.	`isRestartRequired([attributeName])`	isRestartRequired

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.

Example 6-1 Creating a Managed Server

```
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")
disconnect()
exit()
```

The interactive edit session in Example 6-2 changes an Administration Server running in development mode to production mode, and then to secured production mode. Note that your domain must be in production mode to enable secured production mode.

Example 6-2 Changing to Production Mode or Secured Production Mode

```
wls:/offline> 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.setProductionModeEnabled(true)
# Optionally enable secured production mode
```

```
wls:/mydomain/edit !> cd('SecurityConfiguration/mydomain/SecureMode/mydomain')
cmo.setSecureModeEnabled(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()
```

> **Note:**
>
> When using WLST to change the Administration Server from development to production mode, the Java `-Xverify` option (if used) is not changed from `none` to `all` and must be changed manually to `all` to ensure that all classes are verified. In addition, it does not prevent an existing `boot.properties` file from being used when starting the servers.

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

wls:/mydomain/edit/Servers/managed2 !> save()
Saving all your changes ...
Saved all your changes successfully.
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.
Activation completed
wls:/mydomain/edit/Servers/managed2>
```

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 *MBean Reference for Oracle WebLogic Server*.

Example 6-4 Checking the Activation Task

```
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
```

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 *WLST Command Reference for WebLogic Server*.

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 *MBean Reference for Oracle WebLogic Server*.

To access this MBean, use the WLST `getConfigManager` command. See getConfigManager in *WLST Command Reference for WebLogic Server*.

The WLST online script in Example 6-5 connects WLST to a server instance as an administrator, 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

```
connect('adminusername','adminpassword')
user = cmgr.getCurrentEditor()
if user != "weblogic":
    cmgr.undo()
    cmgr.cancelEdit()
cmgr.purgeCompletedActivationTasks()
```

Using WLST Offline to Update an Existing WebLogic Domain

You can update an existing WebLogic domain using WLST offline.

> **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.
>
> The commands in the following table are used to read an existing domain, update the domain as needed, and close the domain in offline mode. During this process, if a connection factory is targeted to a subdeployment, after running the `updateDomain` command and restarting the domain, `default-targeting-enabled` is set to `true` for the connection factory.

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

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

To...	Use this command...	See ...
Open an existing WebLogic domain for update	`readDomain(domainDirName)`	readDomain in *WLST Command Reference for WebLogic Server*
Extend the current WebLogic domain (optional)	`selectTemplate(templateName)` `loadTemplates()`	selectTemplate and loadTemplates in *WLST Command Reference for WebLogic Server*

Table 6-2 (Cont.) Steps for Updating an Existing WebLogic Domain (Offline)

To...	Use this command...	See ...
Modify the WebLogic domain (optional)	Browsing and editing commands	Browsing Information About the Configuration Hierarchy (Offline) Editing a WebLogic Domain (Offline)
Save the WebLogic domain	`updateDomain()`	updateDomain in *WLST Command Reference for WebLogic Server*
Close the WebLogic domain	`closeDomain()`	closeDomain in *WLST Command Reference for WebLogic Server*

Managing Security Data (WLST Online)

You can manage security data, such as authentication providers, using 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*

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

> **Note:**
>
> It is possible to use WLST offline to edit certain types of security data, such as authentication providers. However, we recommend that you use WLST online whenever possible and only use WLST offline to edit security data if required by constraints in your environment .

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

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.

Creating a User

To create a user, invoke the `UserEditorMBean.createUser` method, which is extended by the security realm's `AuthenticationProvider` MBean. See the createUser method of the `UserEditorMBean` in the *MBean Reference for Oracle WebLogic Server*.

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

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

print "Creating a user ..."
atnr=cmo.getSecurityConfiguration().getDefaultRealm().lookupAuthentication
Provider("DefaultAuthenticator")
atnr.createUser('new_user','adminpassword','new_admin')
print "Created user successfully"
```

Adding a User to a Group

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

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 Invoking WLST.

Example 6-7 Adding a User to a Group

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

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

Verifying Whether a User Is a Member of a Group

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

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 Invoking WLST.

Example 6-8 Verifying Whether a User is a Member of a Group

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

Listing Groups to Which a User Belongs

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

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 Invoking WLST.

Example 6-9 Listing Groups to Which a User Belongs

```
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 *MBean Reference for Oracle WebLogic Server*.

Listing Users and Groups in a Security Realm

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

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

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

  See the listGroups operation in the GroupReaderMBean and the listUsers operation in the UserReaderMBean in the *MBean Reference for Oracle WebLogic Server*.
- 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 *MBean Reference for Oracle WebLogic Server*.
- 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 Invoking WLST.

Example 6-10 Listing Users and Groups

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

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

```
    userReader.close(cursor)

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

Changing a Password

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

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 Invoking WLST.

Example 6-11 Changing a Password

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

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

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 *MBean Reference for Oracle WebLogic Server*. For information about collecting lockout statistics and performing lockout operations, see the UserLockoutManagerRuntimeMBean interface in the *MBean Reference for Oracle WebLogic Server*

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.

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

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 Invoking WLST.

Example 6-12 Setting Consecutive Invalid Login Attempts

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

edit()
startEdit()

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

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

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

Unlock a User Account

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

Example 6-13 Unlocking a User Account

```
from weblogic.management.runtime import UserLockoutManagerRuntimeMBean

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

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

Configuring Additional LDAP Authentication Providers

In some cases, such as when installing some Oracle Fusion Middleware products, you must add an additional external LDAP authentication providers 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 *Administering Security for Oracle WebLogic Server*.

Example 6-14 Adding an Authentication Provider

```
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
/'+LDAPProviderName)
cmo.setControlFlag('SUFFICIENT')
cd('/SecurityConfiguration/'+domainName+'/Realms/myrealm/
AuthenticationProviders/'+LDAPProviderName)
cmo.setHost(LDAPHost)
cmo.setPort(LDAPPort)
cmo.setPrincipal(LDAPAdmin)
set("Credential",LDAPAdminPassword)
cmo.setGroupBaseDN(LDAPGroupBase)
cmo.setUserBaseDN(LDAPUserBase)
cmo.setUserNameAttribute(usernameattribute)
cmo.setUserObjectClass('inetOrgPerson')
cd('/SecurityConfiguration/'+domainName+'/Realms/myrealm/AuthenticationProviders
/DefaultAuthenticator')
cmo.setControlFlag('SUFFICIENT')
cd('/SecurityConfiguration/'+domainName+'/Realms/myrealm')
set('AuthenticationProviders',jarray.array([ObjectName('Security:Name=myrealm'
+LDAPProviderName), ObjectName('Security:Name=myrealmDefaultAuthenticator'),
ObjectName('Security:Name=myrealmDefaultIdentityAsserter')], ObjectName))
activate()
```

Table 6-3 lists the class types to specify for each type of Authentication Provider

Table 6-3 Class Types for External LDAP Authentication Providers

Provider	Class Type
Oracle Internet Directory	weblogic.security.providers.authentication.OracleInternetDirectoryAuthenticator
Oracle Virtual Directory	weblogic.security.providers.authentication.OracleVirtualDirectoryAuthenticator
Microsoft AD	weblogic.security.providers.authentication.ActiveDirectoryAuthenticator
OpenLDAP	weblogic.security.providers.authentication.OpenLDAPAuthenticator

Table 6-3 (Cont.) Class Types for External LDAP Authentication Providers

Provider	Class Type
eDirectory	weblogic.security.providers.authentication.NovellAuthenticator
SunOne LDAP	weblogic.security.providers.authentication.IPlanetAuthenticator

Deploying Applications

The process for deploying applications varies depending on whether you use WLST offline or WLST online.

The following topics describe the process:

Using WLST Online to Deploy Applications

When WLST is connected to a domain's Administration Server, use the `deploy` command to deploy applications. (See deploy in *WLST Command Reference for WebLogic Server*.)

The command in Example 6-15 deploys a sample application from the WebLogic Server ExamplesServer domain.

Example 6-15 Deploying Applications

```
# Deploying Applications

deploy("examplesWebApp","C:/Oracle/Middleware/wlserver/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 about using WLST for deploying applications, see Deployment Tools in *Deploying Applications to Oracle WebLogic Server*.

Using WLST Offline to Deploy Applications

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

Table 6-4 Steps for Deploying Applications (Offline)

To...	Use this command...	See ...
Use the Template Builder to create an application template.	not applicable	Creating an Extension Template Using the Domain Template Builder in *Creating Domain Templates Using the Domain Template Builder*
Open an existing WebLogic domain or template	`readDomain(`*`domainDirName`*`)`	readDomain and readTemplate in *WLST Command Reference for WebLogic ServerWebLogic Scripting Tool Command Reference*
Add an extension template to the WebLogic domain	`selectTemplate(templateName)` `loadTemplates()`	selectTemplate and loadTemplates in *WLST Command Reference for WebLogic Server*
Save the WebLogic domain	`updateDomain()`	updateDomain in *WLST Command Reference for WebLogic Server*
Close the WebLogic domain	`closeDomain()`	closeDomain in *WLST Command Reference for WebLogic Server*

For an example of using the `addTemplate` command, see the following sample WLST script, where *`WL_HOME`* refers to the top-level installation directory for WebLogic Server:

`WL_HOME``\common\templates\scripts\wlst\clusterMedRecDomain.py`

7

Updating the Deployment Plan

You can use WebLogic Scripting Tool (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 Using the WebLogic Scripting Tool.

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

> **Note:**
>
> The example commands provided in the following procedure demonstrate how to update and configure the MedRec application, which is installed on your system if you installed the Server Examples.

1. Create a deployment plan for the application.

   See Create a deployment plan in the *Oracle WebLogic Server Administration Console Online 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 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 using the `weblogic.PlanGenerator` utility. You can copy and paste the XPath from the generated plan into your active deployment plan. See weblogic.PlanGenerator Command Line Reference in *Deploying Applications to Oracle WebLogic Server*.

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 is 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 and
   ModuleDescriptor with URI META-INF/weblogic-application.xml.
   Created VariableAssignment with name new_var successfully.
   ```

   c. Set the XPath value for the variable assignment by pasting the XPath value from the deployment plan you generated with `weblogic.PlanGenerator` in Step 4. For example:

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

> **Note:**
>
> To get the correct XPath values for the desired variable assignment, Oracle recommends that you use the `weblogic.PlanGenerator` utility to generate a template deployment plan with empty values. You can then cut or copy the XPath values from the template deployment plan and paste them into the WLST script.

7. Save the deployment plan. For example:

```
plan.save()
```

8
Getting Runtime Information

You can use the WebLogic Scripting Tool (WLST) to retrieve information that WebLogic Server instances produce to describe their run-time state, configure logging, and use the WebLogic Diagnostic Framework.

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. You can access the runtime information using WLST.

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 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.
   -r--   JavaVersion                              1.6.0_21
   -r--   Name                                     AdminServer
   -r--   OSName                                   Windows XP
   -r--   OSVersion                                5.1
   -r--   Type                                     JVMRuntime
   -r--   Uptime                                   409141

   -r-x   preDeregister                            Void :
   ...
   ```

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

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

```
# 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")
```

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 Invoking WLST.

Example 8-2 Monitoring the JVM Heap Size

```
waitTime=180000
THRESHOLD=300000000
uname = "adminusername"
pwd = "adminpassword"
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()
```

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 *MBean Reference for Oracle WebLogic Server*. 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 Invoking WLST.

Example 8-3 Configuring Logging

```
# Connect to the server
connect("adminusername","adminpassword","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()
```

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.

To view 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 exportDiagnosticDatacommand, described in *WLST Command Reference for WebLogic Server*.
- From WLST online, use exportDiagnosticDataFromServer command, described in *WLST Command Reference for WebLogic Server*.

A

WLST Deployment Objects

WLST provides deployment objects that enable you to make changes to a deployment plan and to check the status of a deployment command..

WLSTPlan Object

The `WLSTPlan` object enables you to make changes to an application deployment plan after loading an application using the `loadApplication` command.

loadApplication in *WLST Command Reference for WebLogic Server* describes the command in more detail.

The following table Table A-1describes the `WLSTPlan` object methods that you can use to operate on the deployment plan.

Table A-1 WLSTPlan Object Methods

To operate on the...	Use this method...	To...
Deployment Plan	`DeploymentPlanBean getDeploymentPlan()`	Return the `DeploymentPlanBean` for the current application.
Deployment Plan	`void save() throws FileNotFoundException, ConfigurationException`	Save the deployment plan to a file from which it was read.
Module Descriptors	`ModuleDescriptorBean createModuleDescriptor(String uri, String moduleOverrideName)`	Create a `ModuleDescriptorBean` with the specified `uri` for the `ModuleOverrideBean` `moduleOverrideName`
Module Overrides	`ModuleOverrideBean[] getModuleOverride(String name)`	Return the `ModuleOverrideBean` `name`.
Module Overrides	`ModuleOverrideBean[] getModuleOverrides()`	Return all `ModuleOverrideBean` objects that are available in the deployment plan.
Module Overrides	`void showModuleOverrides()`	Print all of the `ModuleOverrideBean` objects that are available in the deployment plan as name/type pairs.
Variables	`VariableBean createVariable(String name, String value)`	Create a `VariableBean` `name` with this specified value that can override the value in the deployment plan.

Table A-1 (Cont.) WLSTPlan Object Methods

To operate on the...	Use this method...	To...
Variables	`void destroyVariable(String name)`	Destroy the `VariableBean` *name*.
Variables	`VariableBean getVariable(String name)`	Return the `VariableBean` *name*.
Variables	`VariableBean[] getVariables()`	Return all `VariableBean` objects that are available in the deployment plan.
Variables	`void setVariableValue(String name, String value)`	Set the variable *name* to the specified *value*.
Variables	`void showVariables()`	Print all of the `VariableBean` objects in the deployment plan as name/value pairs.
Variable Assignment	`VariableAssignmentBean createVariableAssignment(String name, String moduleOverrideName, String moduleDescriptorUri)`	Create a `VariableAssignmentBean` for the `ModuleDescriptorBean` *moduleDescriptorUri* for the `ModuleOverrideBean` *moduelOverrideName*.
Variable Assignment	`void destroyVariableAssignment(String name, String moduleOverrideName, String moduleDescriptorName)`	Destroy the `VariableAssignmentBean` *name* for the `ModuleDescriptorBean` *moduleOverrideName* for the `ModuleDescriptorBean` *moduleDescriptorName*.
Variable Assignment	`VariableAssignmentBean getVariableAssignment(String name, String moduleOverrideName, String moduleDescriptorName)`	Return the `VariableAssignmentBean` *name* for the `ModuleDescriptorBean` *moduleOverrideName* for the `ModuleDescriptorBean` *moduleDescriptorName*.

WLSTProgress Object

The `WLSTProgress` object enables you to check the status of an executed deployment command.

The `WLSTProgress` object is returned by the following commands (refer to the associated command section in *WLST Command Reference for WebLogic Server*):

Table A-2 describes the `WLSTProgress` object methods that you can use to check the status of the current deployment action.

Table A-2 WLSTProgress Object Methods

Use this method...	To...
`String getCommandType()`	Return the deployment `CommandType` of this event. This command returns one of the following values: `distribute`, `redeploy`, `start`, `stop`, or `undeploy`.
`String getMessage()`	Return information about the status of this event.
`ProgressObject getProgressObject()`	Return the `ProgressObject` that is associated with the current deployment action.
`String getState()`	Retrieve the state of the current deployment action. `CommandType` of this event. This command returns one of the following values: `running`, `completed`, `failed`, or `released` (indicating that the object has been released into production).
`boolean isCompleted()`	Determine if the current deployment action has been completed.
`boolean isFailed()`	Determine if the current deployment action has failed.
`boolean isRunning()`	Determine if the current deployment action is running.
`void printStatus()`	Print the current status of the deployment action, including the command type, the state, additional messages, and so on.

B

FAQs: WLST

Frequently asked questions relating to WLST tend to fall into three categories: General WLST questions, Jython support questions and using WLST questions.

General WLST

General questions about WLST are addressed.

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 WebLogic Server Administration Console?

You can create, configure, and manage WebLogic domains using WLST, command-line utilities, and the WebLogic Server 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.

Jython Support

WLST supports Jython.

What version of Jython is used by WLST?

The WLST scripting environment is based on the Java scripting interpreter, Jython 2.7.1.

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. See http://www.jython.org.

Using WLST

General questions about using WLST are addressed.

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

To connect 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 to `false` to continue execution. This variable defaults to `true`. See WLST Variable Reference in *WLST Command Reference for WebLogic Server*.

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

This is the proper Jython syntax. See http://www.jython.org.

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

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

- Life Cycle Commands in *WLST Command Reference for WebLogic Server*
- Deployment Commands in *WLST Command Reference for WebLogic Server*

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. See connect in *WLST Command Reference for WebLogic Server*.

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 *WLST Command Reference for WebLogic Server*.

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. See Tree Commands in *WLST Command Reference for WebLogic Server*.

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 *WLST Command Reference for WebLogic Server*.

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 through HTTP?

If you are connecting to a WebLogic Server instance through HTTP, ensure that the `TunnelingEnabled` attribute is set to `true` for the WebLogic Server instance. See TunnelingEnabled in *MBean Reference for Oracle WebLogic Server*.

Can I invoke WLST through 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. See 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 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 pointers to the documentation for the custom WLST commands, see Related Documentation in *WLST Command Reference for WebLogic Server*.

You can display help for these commands by entering the `help()`, `help('commandGroup')`, and `help('commandName')` commands on the WLST command line.

Import `wlstModule` for Modularity

Importing `wlstModule` in your WLST scripts allow the script to be run directly or imported without an error.

WLST provides a set of functions and global variables for WLST scripts to use for working with the WebLogic Server MBean servers. Although, basic scripts and command-line allow the use of these functions directly, using this method in your WLST scripts reduces their modularity.

There might be scenarios where data center operations staff are not allowed to access WebLogic Server administrative functions requiring the administrator credentials directly, but instead requires the use of provided WLST scripts. Another scenario, where you are a WLST script developer and have devised a secure way to retrieve the administrator credentials and connect to the production Admin Server. Following is an example:

```
adminUrl = sys.argv[1]
wlUser = ... # secure retrieval of admin user
wlPassword = ... # secure retrieval of admin password

connect(wlUser, wlPassword, adminUrl)
```

This works when you embed it directly in your test script. Now, you want to factor this code out into its own WLST PY function that other scripts use and not copy. To achieve this in Jython, you must create a function in a PY file that your other scripts import. The first step is to move this into a function inside your test script, as shown in the following example:

```
def wlsConnect(adminUrl):
    wlUser = ... # secure retrieval of admin user
    wlPassword = ... # secure retrieval of admin password
    connect(wlUser, wlPassword, adminUrl)
```

The next step is to move the function and others into their own PY file that can be imported into other scripts. Maybe you have created a `wlsAdminUtils.py` file and added your function to it. Now, you can achieve this in all of your scripts that require an administrative connection to the server.

```
import wlsAdminUtils
...
wlAdminUrl = sys.argv[1]
```

```
wlsConnect(wlAdminUrl)
...
```

However, when you try to run your new script, you get an error `Jython NameError: connect` pointing to the call to the WLST connect() function in your `wlsAdminUtils.py-defined wlsConnect` function. This is because the code is now running in a different context, due to the import. To fix and allow the function to run in any context you need to import the `wlstModule` and use it to qualify the name of the WLST functions. Following is an example:

```
import wlstModule as wlst

def wlsConnect(adminUrl):
    wlUser = ... # secure retrieval of admin user
    wlPassword = ... # secure retrieval of admin password
    wlst.connect(wlUser, wlPassword, adminUrl)
```

This ensures that your WLST scripts allow the script to be run directly or imported without an error.

C

WLST Sample Configuration Scripts

You can use these sample WLST scripts as templates to build your own WLST scripts tailored to your environment.

Additional sample scripts are available as part of a WebLogic Server installation. For more information, see WLST Sample Scripts.

Configuring SAML Single Sign On

Use WLST to enable SAML Single Sign On (SSO) on WebLogic Server domains.

When you enable SAML SSO on a WebLogic domain, you need to configure security or authentication providers, SAML 2.0 general services, and, depending on the role of the domain, either Identity Provider services or Service Provider services. For more information on the general process for configuring SAML SSO in WebLogic, see Configuring SAML 2.0 Services in *Administering Security for Oracle WebLogic Server*.

Import Partner Properties

Before you use WLST *offline* to configure SAML SSO, you need to export your federated partners' metadata files, create either an Identity Provider partner properties file or a Service Provider partner properties file, and place both files in the `DOMAIN_HOME/security` directory.

1. Use WSLT *online* to export metadata files from your federated partners. WLST offline does not support exporting metadata.
2. Create a partner properties file :
   - If using WebLogic Server as an Identity Provider, then create a file and name it `saml2sppartner.properties`. Use the following example as a reference. Any properties preceded by `#` are optional.

     ```
     saml2.sp.partners=401kPartner,hmoPartner
     401kPartner.metadata.file=401ksp_metadata.xml

     hmoPartner.metadata.file=hmosp_metadata.xml
     # hmoPartner.enabled=true
     # hmoPartner.description=
     # hmoPartner.mapperClassname=
     # hmoPartner.wantAssertionsSigned=false
     # hmoPartner.timeToLive=100
     # hmoPartner.timeToLiveOffset=50
     # hmoPartner.generateAttributes=false
     # hmoPartner.keyInfoIncluded=false
     # hmoPartner.includeOneTimeUseCondition=false
     ```

   - If using WebLogic Server as a Service Provider, then create a file and name it `saml2idppartner.properties`. Use the following example as a reference. Any properties preceded by `#` are optional.

     ```
     saml2.idp.partners=company1Partner,company2Partner
     company1Partner.description=Company1 IDP Partner
     company1Partner.metadata.file=company1idp_metadata.xml
     ```

```
company1Partner.enabled=true
company1Partner.redirectUris=/company1app/target.jsp,/company1app/
index.jsp

company2Partner.metadata.file=company2idp_metadata.xml
company2Partner.redirectUris=/company2app/target.jsp,/company2app/
welcome.jsp
company2Partner.issuerUri=
company2Partner.enabled=true
company2Partner.virtualUserEnabled=true
#
company2Partner.mapperClassname=com.bea.security.saml2.providers.SAML2Ide
ntityAsserterNameMapper
# company2Partner.wantAssertionsSigned=false
# company2Partner.processAttributes=false
```

3. Save the partner metadata file(s) and the partner properties file in the *DOMAIN_HOME*/security directory.

Sample: Configure WebLogic Server as an Identity Provider Site with SAML SSO

Use this sample WLST script as a starting point to create your own script that configures SAML 2.0 Single Sign On (SSO) on a WebLogic Server instance working as an Identity Provider.

> **Note:**
>
> If you use WLST *offline* to configure SAML SSO, then you need to create a Service Provider partner properties file. This properties file specifies important SAML 2.0 partner metadata that is required by your federated partners. For more information on partner properties files, see Import Partner Properties.

Example C-1 Configure WebLogic Server as an Identity Provider site and enable SAML SSO

Update placeholder text with real values. Placeholder text is enclosed by @ symbols. For example, @admin_username@.

```
def getEnvVar(var):
  val=os.environ.get(var)
  if val==None:
    print "ERROR: Env var ",var, " not set."
    sys.exit(1)
  return val

# Configure SAML2 Credential Mappers
def configSAML2CM():
 cd('/SecurityConfiguration/@domainName@/Realms/@realmName@')
 create('@saml2CMName@',
'com.bea.security.saml2.providers.SAML2CredentialMapper', 'CredentialMapper')
 cd('CredentialMappers')
 cd('@saml2CMName@')
 cmo.setIssuerURI('@url@/company1idp_entityid')
```

```
# Configure SAML1.1 V2 Credential Mappers
def configSAML11V2CM():
 cd('/SecurityConfiguration/@domainName@/Realms/@realmName@')
 create('@samlV2CMName@', 'weblogic.security.providers.saml.SAMLCredentialMapperV2',
'CredentialMapper')
 cd('CredentialMappers')
 cd('@samlV2CMName@')
 cmo.setIssuerURI('@url@/company1idp_saml11issuer')
 cmo.setNameQualifier('company1idp_saml11namequalifier')

# Configure SAML2 SSO Service
def configSSOService(AdminServerName):
 cd('/Server')
 cd(AdminServerName)
 create(AdminServerName, 'SingleSignOnServices')
 cd('SingleSignOnServices')
 cd(AdminServerName)
 cmo.setContactPersonGivenName('company1ContactPersonGivenName')
 cmo.setContactPersonSurName('company1ContactPersonSurName')
 cmo.setContactPersonType('technical')
 cmo.setContactPersonCompany('company1ContactPersonCompany')
 cmo.setContactPersonTelephoneNumber('company1ContactPersonTelephoneNumber')
 cmo.setContactPersonEmailAddress('company1ContactPersonEmailAddress')
 cmo.setOrganizationName('company1OrganizationName')
 cmo.setOrganizationURL('company1OrganizationURL')
 cmo.setEntityID('@url@/company1idp_entityid')
 cmo.setPublishedSiteURL('@url@/saml2')
 cmo.setLoginURL('@url@/loginapp/loginapp.jsp')
 cmo.setIdentityProviderPOSTBindingEnabled(true)
 cmo.setIdentityProviderArtifactBindingEnabled(true)
 cmo.setIdentityProviderRedirectBindingEnabled(true)
 cmo.setIdentityProviderPreferredBinding('HTTP/POST')
 cmo.setSSOSigningKeyAlias('company1IdPSSOSigningKeyAlias')
 ssoSigningKeyPassPhraseEncrypted=encrypt('company1IdPSSOSigningKeyPassPhrase',
'@domainPath@')
 cmo.setSSOSigningKeyPassPhraseEncrypted(ssoSigningKeyPassPhraseEncrypted)

 twoWaySSLEnabled='@twoWaySSLEnabled@'
 if twoWaySSLEnabled == 'true':
         cmo.setTransportLayerSecurityKeyAlias('company1IdPTLSKeyAlias')

transportLayerSecurityKeyPassPhraseEncrypted=encrypt('company1IdPTLSKeyPassPhrase',
'@domainPath@')

cmo.setTransportLayerSecurityKeyPassPhraseEncrypted(transportLayerSecurityKeyPassPhrase
Encrypted)

 cmo.setIdentityProviderEnabled(true)

# Configure SAML1.1 Federation Service
def configFedService(AdminServerName):
 cd('/Server')
 cd(AdminServerName)
 create(AdminServerName, 'FederationServices')
 cd('FederationServices')
 cd(AdminServerName)
 cmo.setSourceSiteURL('@url@/company1idp_saml11')
 cmo.setSigningKeyAlias('company1IdPSSOSigningKeyAlias')
 signingKeyPassPhraseEncrypted=encrypt('company1IdPSSOSigningKeyPassPhrase',
'@domainPath@')
```

```
 cmo.setSigningKeyPassPhraseEncrypted(signingKeyPassPhraseEncrypted)

 isUsingHTTPS='@isUsingHTTPS@'
 twoWaySSLEnabled='@twoWaySSLEnabled@'
 if isUsingHTTPS == 'true':
         cmo.setITSRequiresSSL(true)
         cmo.setARSRequiresSSL(true)
         if twoWaySSLEnabled == 'true':
                 cmo.setARSRequiresTwoWaySSL(true)
 else:
         cmo.setITSRequiresSSL(false)
         cmo.setARSRequiresSSL(false)

 cmo.setSourceSiteEnabled(true)

# Configure SSL
def configSSL(AdminServerName):
 cd('/Servers')
 cd(AdminServerName)
 cmo.setKeyStores('CustomIdentityAndCustomTrust')
 cmo.setCustomTrustKeyStoreFileName('@certsDir@/company1IdPTrust.jks')

customTrustKeyStorePassPhraseEncrypted=encrypt('company1IdPTrustKeyStorePassPhras
e', '@domainPath@')

cmo.setCustomTrustKeyStorePassPhraseEncrypted(customTrustKeyStorePassPhraseEncryp
ted)
 cmo.setCustomIdentityKeyStoreFileName('@certsDir@/company1IdPIdentity.jks')

customIdentityKeyStorePassPhraseEncrypted=encrypt('company1IdPIdentityKeyStorePas
sPhrase', '@domainPath@')

cmo.setCustomIdentityKeyStorePassPhraseEncrypted(customIdentityKeyStorePassPhrase
Encrypted)

 create(AdminServerName, 'SSL')
 cd('/Servers/' + AdminServerName + '/SSL')
 cd(AdminServerName)
 cmo.setEnabled(true)
 cmo.setListenPort(int('@sport@'))
 cmo.setTwoWaySSLEnabled(Boolean('@twoWaySSLEnabled@'))
 cmo.setClientCertificateEnforced(Boolean('@clientCertificateEnforced@'))
 cmo.setHostnameVerificationIgnored(false)
 cmo.setServerPrivateKeyAlias('company1IdPServerKeyAlias')
 serverPrivateKeyPassPhraseEncrypted=encrypt('company1IdPServerKeyPassPhrase',
'@domainPath@')
 cmo.setServerPrivateKeyPassPhraseEncrypted(serverPrivateKeyPassPhraseEncrypted)

# Optional: Create cluster

# Create IDP domain

readDomain('@domainPath@')
configSAML2CM()
configSSOService('@adminServerName@')

# configSAML11V2CM()
configFedService('@adminServerName@')
configSSL('@adminServerName@')

updateDomain()
```

```
closeDomain()
print 'Domain Updated with Identity Provider configured'

exit()
```

Sample: Configure WebLogic Server as a Service Provider Site with SAML SSO

Use this sample WLST script as a starting point to create your own script that configures SAML 2.0 Single Sign On (SSO) on a WebLogic Server instance working as a Service Provider.

> **Note:**
>
> If you use WLST *offline* to configure SAML SSO, then you need to create an Identity Provider partner properties file. This properties file specifies important SAML 2.0 partner metadata that is required by your federated partners. For more information on partner properties files, see Import Partner Properties.

Example C-2 Configure WebLogic Server as a Service Provider site and enable SAML SSO

Update placeholder text with real values. Placeholder text is enclosed by `@` symbols. For example, `@admin_username@`.

```
def getEnvVar(var):
  val=os.environ.get(var)
  if val==None:
    print "ERROR: Env var ",var, " not set."
    sys.exit(1)
  return val

# Create domain
def createDomain(domainName, adminServerName):
  readTemplate('@templateJar@')
  set('Name', domainName)
  setOption('DomainName', domainName)
  cd('/Servers/AdminServer')
  set('ListenPort', '@admin_port@')
  set('Name', adminServerName)
  cd('/Security/' + domainName + '/User/weblogic')
  cmo.setName('@admin_username@')
  cmo.setPassword('@admin_password@')
  setOption('OverwriteDomain', 'true')
  writeDomain('@domainPath@')
  closeTemplate()
  print 'Domain Created'

# Create a cluster
def createCluster(clusterName):
  cd('/')
  cl=create(clusterName, 'Cluster')
  cluster_type='@cluster_type@'
  number_of_ms=int('@number_of_ms@')
  managed_server_name_base='@managed_server_name_base@'
```

```
  managed_server_name_base_svc='@managed_server_name_base_svc@'

  if cluster_type == "CONFIGURED":
    for index in range(0, number_of_ms):
      cd('/')
      msIndex = index+1
      name = managed_server_name_base + msIndex
      name_svc = managed_server_name_base_svc + msIndex
      create(name, 'Server')
      cd('/Servers/' + name + '/')
      print('managed server name is ' + name)
      set('ListenPort', '@server_port@')
      set('NumOfRetriesBeforeMSIMode', 0)
      set('RetryIntervalBeforeMSIMode', 1)
      set('Cluster', clusterName)

  else:
    print('Configuring Dynamic Cluster ' +  clusterName)
    templateName = '@cluster_name@-template'
    print('Creating Server Template: ' + templateName)
    st1=create(templateName, 'ServerTemplate')
    print('Done creating Server Template: ' + templateName)
    cd('/ServerTemplates/' + templateName)
    cmo.setListenPort('@server_port@')
    cmo.setCluster(cl)
    print('Done setting attributes for Server Template: ' + templateName);
    cd('/Clusters/' + clusterName)
    create(clusterName, 'DynamicServers')
    cd('DynamicServers/' + clusterName)
    set('ServerTemplate', st1)
    set('ServerNamePrefix', managed_server_name_base)
    set('DynamicClusterSize', number_of_ms)
    set('MaxDynamicClusterSize', number_of_ms)
    set('CalculatedListenPorts', false)

  print('Done setting attributes for Dynamic Cluster: ' + clusterName);

# Configure SAML Authentication Provider
def configSAMLAtn():
  cd('/SecurityConfiguration/@domainName@/Realms/@realmName@')
  samlatn = create('@samlAtnName@',
'weblogic.security.providers.saml.SAMLAuthenticator', 'AuthenticationProvider')
  samlatn.setControlFlag('SUFFICIENT')

# Configure SAML2 Identity Asserter
def configSAML2IA():
  cd('/SecurityConfiguration/@domainName@/Realms/@realmName@')
  create('@saml2IAName@',
'com.bea.security.saml2.providers.SAML2IdentityAsserter',
'AuthenticationProvider')
  cd('AuthenticationProvider')
  cd('@saml2IAName@')
  # cmo.setReplicatedCacheEnabled(Boolean('@replicatedCacheEnabled@'))

def reConfigDefaultAtn():
  cd('/SecurityConfiguration/@domainName@/Realms/@realmName@')
  delete('DefaultAuthenticator', 'AuthenticationProvider')
  delete('DefaultIdentityAsserter','AuthenticationProvider')
  defaultAtn=create('DefaultAuthenticator',
'weblogic.security.providers.authentication.DefaultAuthenticator',
'AuthenticationProvider')
```

```
  defaultAtn.setControlFlag('REQUIRED')
  create('DefaultIdentityAsserter',
'weblogic.security.providers.authentication.DefaultIdentityAsserter',
'AuthenticationProvider')

# Configure SAML1.1 V2 Identity Asserter
def configSAML11V2IA():
  cd('/SecurityConfiguration/@domainName@/Realms/@realmName@')
  create('@saml11IAName@', 'weblogic.security.providers.saml.SAMLIdentityAsserterV2',
'AuthenticationProvider')
  cd('AuthenticationProvider')
  cd('@saml11IAName@')

# Configure SAML2 SSO Service
def configSSOService(AdminServerName):
 cd('/Server')
 cd(AdminServerName)
 create(AdminServerName, 'SingleSignOnServices')
 cd('SingleSignOnServices')
 cd(AdminServerName)
 cmo.setContactPersonGivenName('401kContactPersonGivenName')
 cmo.setContactPersonSurName('401kContactPersonSurName')
 cmo.setContactPersonType('technical')
 cmo.setContactPersonCompany('401kContactPersonCompany')
 cmo.setContactPersonTelephoneNumber('401kContactPersonTelephoneNumber')
 cmo.setContactPersonEmailAddress('401kContactPersonEmailAddress')
 cmo.setOrganizationName('401kOrganizationName')
 cmo.setOrganizationURL('401kOrganizationURL')
 cmo.setEntityID('@url@/401ksp_entityid')
 cmo.setPublishedSiteURL('@url@/saml2')
 cmo.setServiceProviderPOSTBindingEnabled(true)
 cmo.setServiceProviderArtifactBindingEnabled(true)
 cmo.setServiceProviderPreferredBinding('HTTP/POST')
 cmo.setSSOSigningKeyAlias('401kSPSSOSigningKeyAlias')
 ssoSigningKeyPassPhraseEncrypted=encrypt('401kSPSSOSigningKeyPassPhrase',
'@domainPath@')
 cmo.setSSOSigningKeyPassPhraseEncrypted(ssoSigningKeyPassPhraseEncrypted)

 twoWaySSLEnabled='@twoWaySSLEnabled@'
 if twoWaySSLEnabled == 'true':
        cmo.setTransportLayerSecurityKeyAlias('401kSPTLSKeyAlias')

transportLayerSecurityKeyPassPhraseEncrypted=encrypt('401kSPTLSKeyPassPhrase',
'@domainPath@')

cmo.setTransportLayerSecurityKeyPassPhraseEncrypted(transportLayerSecurityKeyPassPhrase
Encrypted)

 cmo.setServiceProviderEnabled(true)
 print "SP Service configured."

# Configure SAML1.1 Federation Service
def configFedService(AdminServerName):
 cd('/Server')
 cd(AdminServerName)
 create(AdminServerName, 'FederationServices')
 cd('FederationServices')
 cd(AdminServerName)
 isUsingHTTPS='@isUsingHTTPS@'
 if isUsingHTTPS == 'true':
        cmo.setACSRequiresSSL(true)
```

```
 else:
         cmo.setACSRequiresSSL(false)
 cmo.setDestinationSiteEnabled(true)

# Configure Keystores
def configSSL(AdminServerName):
 cd('/Servers')
 cd(AdminServerName)
 cmo.setKeyStores('CustomIdentityAndCustomTrust')
 cmo.setCustomTrustKeyStoreFileName('@certsDir@/401kSPTrust.jks')
 customTrustKeyStorePassPhraseEncrypted=encrypt('401kSPTrustKeyStorePassPhrase',
'@domainPath@')

cmo.setCustomTrustKeyStorePassPhraseEncrypted(customTrustKeyStorePassPhraseEncryp
ted)
 cmo.setCustomIdentityKeyStoreFileName('@certsDir@/401kSPIdentity.jks')

customIdentityKeyStorePassPhraseEncrypted=encrypt('401kSPIdentityKeyStorePassPhra
se', '@domainPath@')

cmo.setCustomIdentityKeyStorePassPhraseEncrypted(customIdentityKeyStorePassPhrase
Encrypted)

 create(AdminServerName, 'SSL')
 cd('/Servers/' + AdminServerName + '/SSL')
 cd(AdminServerName)
 cmo.setEnabled(true)
 cmo.setListenPort(int('@sport@'))
 cmo.setTwoWaySSLEnabled(Boolean('@twoWaySSLEnabled@'))
 cmo.setClientCertificateEnforced(Boolean('@clientCertificateEnforced@'))
 cmo.setHostnameVerificationIgnored(false)
 cmo.setServerPrivateKeyAlias('401kSPServerKeyAlias')
 serverPrivateKeyPassPhraseEncrypted=encrypt('401kSPServerKeyPassPhrase',
'@domainPath@')
 cmo.setServerPrivateKeyPassPhraseEncrypted(serverPrivateKeyPassPhraseEncrypted)

# Open the existing domain and configure SP
createDomain = '@createDomain@'
createCluster = '@createCluster@'

if createDomain == 'true':
  createDomain('@domainName@', '@adminServerName@')

readDomain('@domainPath@')
if createCluster == 'true':
  createCluster('@clusterName@')

configSAMLAtn()
configSAML2IA()
reConfigDefaultAtn()
configSSOService('@adminServerName@')
configFedService('@adminServerName@')
configSSL('@adminServerName@')

updateDomain()
closeDomain()
print 'Domain Updated with Service Provider Configured'

exit()
```

D

WLST Deprecated Features

Some WLST features are deprecated.

Implicit Exports

Implicit imports are deprecated, beginning in Release 12.2.1.2.

Currently, by default, when WLST is started, the weblogic.jar is automatically added to the Jython sys.path so that all weblogic-related classes referenced in weblogic.jar are loaded into the WLST namespace. You can just reference these weblogic classes in your py script by calling weblogic.*class* directly. There is no need to import those weblogic classes explicitly by calling them using `from weblogic import class`. This provides a convenient way to reference weblogic classes in your py script.

However, loading weblogic classes into the WLST namespace during WLST startup adds execution time during startup and can cause performance issues.

Currently, you can disable auto-loading during the WLST startup by setting the following system property to true:

```
python.cachedir.skip
```

In a future release, WLST will disable weblogic class auto-loading during WLST startup. At that point, you will need to explicitly import classes using from weblogic import class. For example:

```
from weblogic.security.service import EJBResourceejbRes = EJBResource('DDPoliciesEar',
'DDPolinEarMiniAppBean.jar', 'DDRolesAndPolicies', 'getSubject', 'Remote', None)
```