Oracle® Fusion Middleware

WLST Command Reference for Oracle WebLogic Server

14c (14.1.1.0.0)
F18293-08
March 2023

Oracle Fusion Middleware WLST Command Reference for Oracle WebLogic Server, 14c (14.1.1.0.0)

F18293-08

Contents

Preface

This document describes all of the commands that are available to use with the WebLogic Scripting Tool (WLST).

Audience

This document is written for WebLogic Server administrators and operators who deploy Java EE applications using the Java Platform, Enterprise Edition (Java EE) from Oracle. 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 how to use the WebLogic Scripting Tool, refer to *Understanding the WebLogic Scripting Tool*.

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.
- *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.

New and Changed WebLogic Server Features

See *What's New in Oracle WebLogic Server*.

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

WebLogic Server WLST Online and Offline Command Reference

The WebLogic Scripting Tool (WLST) is a Jython-based command-line scripting environment that you can use to create, manage, and monitor WebLogic domains. See a summary of the WLST commands for Oracle WebLogic Server, listed in alphabetical order, along with an indication whether a command can be used in offline mode, online mode, or both.

This chapter includes the following sections:

- WebLogic Server WLST Command Summary, Alphabetically By Command
- WebLogic Server WLST Online Command Summary
- WebLogic Server WLST Offline Command Summary

Note:

You can list a summary of all online and offline commands from the command-line using the following commands, respectively:

```
help("online")
help("offline")
```

WebLogic Server WLST Command Summary, Alphabetically By Command

See an alphabetical listing of every WebLogic Server WLST command. The following table includes a summary of each WebLogic Server WLST command. For each command, this table indicates whether the command can be used in onine mode, offline mode, or both.

Table 1-1 WebLogic Server WLST Command Summary

This command...	Enables you to...	Use with WLST...
activate	Activate changes saved during the current editing session but not yet deployed.	Online
activateDebugPatch	Activate a debug patch on specified targets.	Online
addHelpCommand	Add new command help for a command to an existing command group. Once added to the group, the command (along with a brief description) is displayed in the command list for the group when you enter the `help('commandGroup')` command.	Online or Offline
addHelpCommandGroup	Add a new help command group to those shown by the WLST `help()` command.	Online or Offline

Table 1-1 (Cont.) WebLogic Server WLST Command Summary

This command...	Enables you to...	Use with WLST...
addListener	Add a JMX listener to the specified MBean.	Online
addStartupGroup	Add a new server startup group.	Offline
addTemplate	Extend the current WebLogic domain using an application or service extension template.	Offline
assign	Assign resources to one or more destinations.	Offline
cancelEdit	Cancel an edit session, release the edit lock, and discard all unsaved changes. This operation can be called by any user with administrator privileges, even if the user did not start the edit session.	Online
captureAndSaveDiagnosticImage	Capture a diagnostic image and downloads it to the client.	Online
cd	Navigate the hierarchy of configuration or runtime beans.	Online or Offline
clone	Clone the server object.	Offline
closeDomain	Close the current WebLogic domain.	Offline
closestore	Close a store.	Offline
closeTemplate	Close the current domain template.	Offline
compactstore	Compact and defragments the space occupied by a file store.	Offline
configToScript	Convert an existing server configuration (`config` directory) to an executable WLST script.	Online or Offline
connect	Connect WLST to a WebLogic Server instance.	Online or Offline
create	Create a configuration bean of the specified type for the current bean.	Online or Offline
createDomain	Create a WebLogic domain using the specified template.	Offline
createEditSession	Create a new edit session.	Online
createSystemResourceControl	Create a diagnostics resource from an external descriptor file.	Online
currentTree	Return the current location in the hierarchy.	Online
custom	Navigate to the root of custom MBeans that are registered in the Runtime MBean Server.	Online
deactivateAllDebugPatches	Deactivate all debug patches on specified targets.	Online
deactivateDebugPatches	Deactivate debug patches on specified targets.	Online
delete	Delete an instance of a configuration bean of the specified type for the current configuration bean.	Online or Offline
deleteFEHost	Delete the Frontend host for a domain.	Offline
deleteStartupGroup	Delete a server startup group.	Offline
deploy	Deploy an application to a WebLogic Server instance.	Online
destroySystemResourceControl	Destroy a Diagnostics system resource control.	Online

Table 1-1 (Cont.) WebLogic Server WLST Command Summary

This command...	Enables you to...	Use with WLST...
destroyEditSession	Remove the specified edit session.	Online
disableSystemResource	Deactivate a diagnostic system resource control that is activated on a server instance.	Online
disconnect	Disconnect WLST from a WebLogic Server instance.	Online
distributeApplication	Copy the deployment bundle to the specified targets.	Online
domainConfig	Navigate to the last MBean to which you navigated in the domain configuration hierarchy or to the root of the hierarchy, `DomainMBean`.	Online
domainCustom	Navigate to the tree of custom MBeans that are registered in the Domain Runtime MBean Server.	Online
domainRuntime	Navigate to the last MBean to which you navigated in the domain runtime hierarchy or to the root of the hierarchy, `DomainRuntimeMBean`.	Online
dumpDiagnosticData	Dump the diagnostics data from a harvester to a local file.	Online
dumpStack	Display a stack trace from the last exception that occurred while performing a WLST action, and reset the stack trace.	Online or Offline
dumpstore	Dump store contents in human-readable format to an XML file.	Offline
dumpVariables	Display all variables used by WLST, including their name and value.	Online or Offline
edit	Navigate to the last MBean to which you navigated in the configuration edit MBean hierarchy or to the root of the hierarchy, `DomainMBean`.	Online
editCustom	Navigate to the root of custom MBeans.	Online
encrypt	Encrypt the specified string.	Online
enableOverwriteComponentChanges	Overwrite changes to all system components during activation.	Online
enableSystemResource	Activate a diagnostic system resource on a server instance.	Online
exit	Exit WLST from the user session and close the scripting shell.	Online or Offline
exportDiagnosticData	Execute a query against the specified log file.	Offline
exportDiagnosticDataFromServer	Execute a query on the server side and retrieves the exported WebLogic Diagnostic Framework (WLDF) data.	Online
exportHarvestedTimeSeriesData	Export the harvested metric data within the specified interval in CSV format.	Online
exportHarvestedTimeSeriesDataOffline	Export the harvested metric data in offline mode within the specified interval in CSV format.	Offline
find	Find MBeans and attributes in the current hierarchy.	Online
get	Return the value of the specified attribute.	Online or Offline
getActivationTask	Return the latest `ActivationTask` MBean on which a user can get status.	Online
getAvailableCapturedImages	Return a list of the previously captured diagnostic images.	Online

Table 1-1 (Cont.) WebLogic Server WLST Command Summary

This command...	Enables you to...	Use with WLST...
getAvailableDiagnosticDataAccessorNames	Get the diagnostic data accessor names currently available on a server.	Online
getConfigManager	Return the latest `ConfigurationManagerBean` MBean which manages the change process.	Online
getDatabaseDefaults	Connect to the database to retrieve schema information.	Offline
getFEHostURL	Retrieve the settings for a domain's Frontend Host.	Offline
getMBean	Return the MBean by browsing to the specified path.	Online
getMBI	Return the `MBeanInfo` for the specified `MBeanType` or the `cmo` variable.	Online
getNodeManagerHome	Get the Node Manager home.	Offline
getNodeManagerType	Get the Node Manager type.	Offline
getNodeManagerUpgradeOverwriteDefault	Get the value of the Node Manager upgrade overwrite default flag.	Offline
getNodeManagerUpgradeType	Get the Node Manager upgrade type used for Node Manager upgrade.	Offline
getOldNodeManagerHome	Get the old Node Manager home used for Node Manager upgrade.	Offline
getopenstores	Return a list of opened stores (for script access).	Offline
getstoreconns	Return a list of connections in the specified store (for script access).	Offline
getPath	Return the MBean path for the specified MBean instance.	Online
getServerGroups	Retrieve a list of the server groups of which the specified server is a member.	Offline
getStartupGroup	Retrieve the server startup group.	Offline
getTopologyProfile	Return the domain topology profile for a domain.	Offline
getWLDM	Return the WebLogic `DeploymentManager` object.	Online
invoke	Invoke a management operation on the current configuration bean.	Online
isRestartRequired	Determine whether a server restart is required.	Online
jndi	Navigate to the JNDI tree for the server to which WLST is currently connected.	Online
listApplications	List all applications that are currently deployed in the domain.	Online
listChildTypes	List all the children MBeans that can be created or deleted for the `cmo`.	Online
listDebugPatches	List active and available debug patches on specified targets.	Online
listDebugPatchTasks	List debug patch tasks from specified targets.	Online
listServerGroups	Retrieve a map of the config-groups.xml server groups that are user-expandable.	Offline
liststore	List store names, open stores or connections in a store.	Offline
listSystemResourceControls	List the diagnostic system resources that are currently deployed on a server instance.	Online

Table 1-1 (Cont.) WebLogic Server WLST Command Summary

This command...	Enables you to...	Use with WLST...
loadApplication	Load an application and deployment plan into memory.	Online or Offline
loadDB	Load SQL files into a database.	Offline
loadProperties	Load property values from a file.	Online or Offline
loadTemplates	Load all the selected templates using select template.	Offline
lookup	Look up the specified MBean.	Online
ls	List all child beans and/or attributes for the current configuration or runtime bean.	Online or Offline
man	Display help from `MBeanInfo` for the current MBean or its specified attribute.	Online
mergeDiagnosticData	Merge a set of data files.	Online
migrate	Migrate services to a target server within a cluster.	Online
nm	Determine whether WLST is connected to Node Manager.	Online
nmConnect	Connect WLST to Node Manager to establish a session.	Online or Offline
nmDisconnect	Disconnect WLST from a Node Manager session.	Online or Offline
nmEnroll	Enroll the machine on which WLST is currently running.	Online
nmExecScript	Execute the named script using the connected Node Manager.	Online
nmGenBootStartupProps	Generate the Node Manager property files, `boot.properties` and `startup.properties`, for the specified server.	Online
nmKill	Kill the specified server instance that was started with Node Manager.	Online or Offline
nmLog	Return the Node Manager log.	Online or Offline
nmRestart	Restart the Node Manager instance.	Online
nmServerLog	Return the server output log of the server that was started with Node Manager.	Online or Offline
nmServerStatus	Return the status of the server that was started with Node Manager.	Online or Offline
nmSoftRestart	Restart the specified System Component server instance.	Online or Offline
nmStart	Start a server in the current domain using Node Manager.	Online or Offline
nmVersion	Return the Node Manager server version.	Online or Offline
openfilestore	Open a file store.	Offline
openjdbcstore	Open a JDBC store.	Offline
prompt	Toggle the display of path information at the prompt.	Online or Offline

Table 1-1 (Cont.) WebLogic Server WLST Command Summary

This command...	Enables you to...	Use with WLST...
pullComponentChanges	Pull the configuration changes of a system component from the remote node.	Online
purgeCapturedImages	Purge the diagnostic image files on the server as per the age criteria specified.	Online
purgeDebugPatchTasks	Purge debug patch tasks from specified targets.	Online
pwd	Display the current location in the configuration or runtime bean hierarchy.	Online or Offline
readDomain	Open an existing WebLogic domain for updating.	Offline
readDomainForUpgrade	Open an existing domain for reconfiguration.	Offline
readTemplate	Open an existing domain template for WebLogic domain creation.	Offline
readTemplateForUpdate	Open an existing domain template for template update.	Offline
redeploy	Reload classes and redeploy a previously deployed application.	Online
redirect	Redirect WLST output to the specified filename.	Online or Offline
removeListener	Remove a listener that was previously defined.	Online
resolve	Detect any external modification or conflict, and resolve them.	Online
resume	Resume a server instance that is suspended or in `ADMIN` state.	Online
resync	Resynchronize the configuration files of the specified system component.	Online
resyncAll	Resynchronize the configuration files of all system components.	Online
rolloutApplications	Roll out updates to specified applications on the targeted servers without interrupting service.	Online
rolloutJavaHome	Roll out a new Java Home for targeted servers without interrupting service.	Online
rolloutOracleHome	Roll out a patched Oracle Home to the targeted server without interrupting service.	Online
rollingRestart	Initiate a rolling restart of the targeted servers without interrupting the service.	Online
rolloutUpdate	Roll out updates to targeted servers without interrupting service.	Online
save	Save the edits that have been made but have not yet been saved.	Online
saveDiagnosticImageCaptureFile	Download the specified diagnostic image capture.	Online
saveDiagnosticImageCaptureEntryFile	Download a specific entry from the diagnostic image capture.	Online
scaleDown	Decrease the number of running dynamic servers in the specified dynamic cluster.	Online
scaleUp	Increase the number of running dynamic servers for the specified dynamic cluster.	Online
selectCustomTemplate	Select an existing custom domain template or application template for creating a domain.	Offline

Table 1-1 (Cont.) WebLogic Server WLST Command Summary

This command...	Enables you to...	Use with WLST...
selectTemplate	Select an existing domain template or application template for creating a domain.	Offline
serverConfig	Navigate to the last MBean to which you navigated in the configuration MBean hierarchy or to the root of the hierarchy, `DomainMBean`.	Online
serverRuntime	Navigate to the last MBean to which you navigated in the runtime MBean hierarchy or to the root of the hierarchy, `ServerRuntimeMBean`.	Online
set	Set the specified attribute value for the current configuration bean.	Online or Offline
setFEHostURL	Set the plain and SSL URLs for a domain's Frontend Host and specifies which is the default.	Offline
setOption	Set options related to a WebLogic domain creation or update.	Offline
setServerGroups	Set the server groups for the specified server.	Online or Offline
setSharedSecretStoreWithPassword	Set the shared secret store and password.	Offline
setShowLSResult	Specify whether `ls()` should log its output to stdout.	Online or Offline
setStartupGroup	Set the server startup group.	Offline
setTopologyProfile	Set the domain topology profile.	Offline
showAvailableTemplates	Display all currently available templates for loading.	Offline
showChanges	Show the changes made by the current user during the current edit session.	Online
showComponentChanges	Show the changes to the configuration of the specified system component on the remote node.	Online
showDebugPatchInfo	Display details about a debug patch on specified targets.	Online
showEditSession	Show information about the specified edit sessions.	Online
showListeners	Show all listeners that are currently defined.	Online
showTemplates	Display all currently selected and loaded templates.	Offline
shutdown	Gracefully shut down a running server instance, cluster, or system component.	Online
softRestart	Restart a system component server instance.	Online
start	Start a Managed Server instance, cluster, or system component using Node Manager.	Online
startApplication	Start an application, making it available to users.	Online
startEdit	Start a configuration edit session on behalf of the currently connected user.	Online
startNodeManager	Start Node Manager at default port (5556).	Online or Offline
startRecording	Record all user interactions with WLST; useful for capturing commands to replay.	Online or Offline

Table 1-1 (Cont.) WebLogic Server WLST Command Summary

This command...	Enables you to...	Use with WLST...
startServer	Start the Administration Server.	Online or Offline
state	Returns a map of servers or clusters and their state using Node Manager.	Online
stopApplication	Stop an application, making it un available to users.	Online
stopEdit	Stop the current edit session, release the edit lock, and discard unsaved changes.	Online
stopNodeManager	Stop Node Manager.	Online or Offline
stopRecording	Stop recording WLST commands.	Online or Offline
stopRedirect	Stop the redirection of WLST output to a file.	Online or Offline
storeUserConfig	Create a user configuration file and an associated key file.	Online
suspend	Suspend a running server.	Online
threadDump	Display a thread dump for the specified server.	Online or Offline
undeploy	Undeploy an application from the specified servers.	Online
updateApplication	Update an application configuration using a new deployment plan.	Online
updateDomain	Update and save the current domain.	Offline
unassign	Unassign applications or services from one or more destinations.	Offline
undeploy	Undeploy an application from the specified servers.	Online
undo	Revert all unsaved or unactivated edits.	Online
unselectCustomTemplate	Deselect the selected custom domain template or application template.	Offline
unselectTemplate	Deselect the selected domain template or application template.	Offline
validate	Validate the changes that have been made but have not yet been saved.	Online
validateConfig	Validate a domain configuration.	Offline
viewMBean	Display information about an MBean, such as the attribute names and values, and operations.	Online
writeDomain	Write the domain configuration information to the specified directory.	Offline
writeIniFile	Convert WLST definitions and method declarations to a Python (`.py`) file.	Online or Offline
writeTemplate	Write the domain configuration information to the specified domain template.	Online or Offline

WebLogic Server WLST Online Command Summary

See a summary of the WLST online commands for Oracle WebLogic Server, listed in alphabetical order.

Table 1-2 WebLogic Server WLST Online Command Summary

This command...	Enables you to...
activate	Activate changes saved during the current editing session but not yet deployed.
activateDebugPatch	Activate a debug patch on specified targets.
addHelpCommand	Adds new command help for a command to an existing command group. Once added to the group, the command (along with a brief description) is displayed in the command list for the group when you enter the `help('commandGroup')` command.
addHelpCommandGroup	Adds a new help command group to those shown by the WLST `help()` command, and specifies the resource bundle in which the help information is defined for the group.
addListener	Add a JMX listener to the specified MBean.
cancelEdit	Cancel an edit session, release the edit lock, and discard all unsaved changes. This operation can be called by any user with administrator privileges, even if the user did not start the edit session.
captureAndSaveDiagnosticImage	Capture a diagnostic image and downloads it to the client.
cd	Navigate the hierarchy of configuration or runtime beans.
configToScript	Convert an existing server configuration (`config` directory) to an executable WLST script.
connect	Connect WLST to a WebLogic Server instance.
create	Create a configuration bean of the specified type for the current bean.
createEditSession	Create a new edit session.
createSystemResourceControl	Creates a diagnostics resource from an external descriptor file.
currentTree	Return the current tree location.
custom	Navigate to the root of custom MBeans that are registered in the Runtime MBean Server.
deactivateAllDebugPatches	Deactivate all debug patches on specified targets.
deactivateDebugPatches	Deactivate debug patches on specified targets.
delete	Delete an instance of a configuration bean of the specified type for the current configuration bean.
deploy	Deploy an application to a WebLogic Server instance.
destroySystemResourceControl	Destroys a Diagnostics system resource control.
destroyEditSession	Remove the specified edit session.
disableSystemResource	Deactivate a diagnostic system resource control that is activated on a server instance.
disconnect	Disconnect WLST from a WebLogic Server instance.
distributeApplication	Copy the deployment bundle to the specified targets.
domainConfig	Navigate to the last MBean to which you navigated in the domain configuration hierarchy or to the root of the hierarchy, `DomainMBean`.
domainCustom	Navigate to the tree of custom MBeans that are registered in the Domain Runtime MBean Server.
domainRuntime	Navigate to the last MBean to which you navigated in the domain runtime hierarchy or to the root of the hierarchy, `DomainRuntimeMBean`.

Table 1-2 (Cont.) WebLogic Server WLST Online Command Summary

This command...	Enables you to...
dumpDiagnosticData	Dumps the diagnostics data from a harvester to a local file.
dumpStack	Display a stack trace from the last exception that occurred, and reset the trace.
dumpVariables	Display all variables used by WLST, including their name and value.
edit	Navigate to the last MBean to which you navigated in the configuration edit MBean hierarchy or to the root of the hierarchy, `DomainMBean`.
editCustom	Navigate to the root of custom MBeans.
enableOverwriteComponentChanges	Overwrite changes to all system components during activation.
enableSystemResource	Activate a diagnostic system resource on a server instance
encrypt	Encrypt the specified string.
exit	Exit WLST from the interactive session and close the scripting shell.
exportDiagnosticDataFromServer	Execute a query on the server side and retrieve the exported WebLogic Diagnostic Framework (WLDF) data.
exportHarvestedTimeSeriesData	Export the harvested metric data within the specified interval in CSV format.
find	Find MBeans and attributes in the current hierarchy.
get	Return the value of the specified attribute.
getActivationTask	Return the latest `ActivationTask` MBean on which a user can get status.
getAvailableCapturedImages	Return a list of the previously captured diagnostic images.
getAvailableDiagnosticDataAccessorNames	Get the diagnostic data accessor names currently available on a server.
getConfigManager	Return the latest `ConfigurationManagerBean` MBean which manages the change process.
getMBean	Return the MBean by browsing to the specified path.
getMBI	Return the `MBeanInfo` for the specified `MBeanType` or the `cmo` variable.
getPath	Return the MBean path for the specified MBean instance.
getWLDM	Return the WebLogic `DeploymentManager` object.
invoke	Invoke a management operation on the current configuration bean.
isRestartRequired	Determine whether a server restart is required.
jndi	Navigates to the JNDI tree for the server to which WLST is currently connected.
listApplications	List all applications that are currently deployed in the domain.
listChildTypes	List all the children MBeans that can be created or deleted for the `cmo`.
listDebugPatches	List active and available debug patches on specified targets.
listDebugPatchTasks	List debug patch tasks from specified targets.
listSystemResourceControls	List the diagnostic system resources that are currently deployed on a server instance.
loadApplication	Load an application and deployment plan into memory.
loadProperties	Load property values from a file.
lookup	Look up the specified MBean.

Table 1-2 (Cont.) WebLogic Server WLST Online Command Summary

This command...	Enables you to...
ls	List all child beans and/or attributes for the current configuration or runtime bean.
man	Display help from `MBeanInfo` for the current MBean or its specified attribute.
mergeDiagnosticData	Merges a set of data files.
migrate	Migrate services to a target server within a cluster.
nm	Determine whether WLST is connected to Node Manager.
nmConnect	Connect WLST to Node Manager to establish a session.
nmDisconnect	Disconnect WLST from a Node Manager session.
nmEnroll	Enroll the machine on which WLST is currently running.
nmExecScript	Execute the named script using the connected Node Manager.
nmGenBootStartupProps	Generates the Node Manager property files, `boot.properties` and `startup.properties`, for the specified server.
nmKill	Kill the specified server instance that was started with Node Manager.
nmLog	Return the Node Manager log.
nmRestart	Restart the Node Manager instance.
nmServerLog	Return the server output log of the server that was started with Node Manager.
nmServerStatus	Return the status of the server that was started with Node Manager.
nmSoftRestart	Restart the specified System Component server instance.
nmStart	Start a server in the current domain using Node Manager.
nmVersion	Return the Node Manager server version.
prompt	Toggle the display of path information at the prompt.
pullComponentChanges	Pull the configuration changes of a system component from the remote node.
purgeCapturedImages	Purge the diagnostic image files on the server as per the age criteria specified.
purgeDebugPatchTasks	Purge debug patch tasks from specified targets.
pwd	Display the current location in the configuration or runtime bean hierarchy.
redeploy	Reload classes and redeploy a previously deployed application.
redirect	Redirect WLST output to the specified filename.
removeListener	Remove a listener that was previously defined.
resolve	Detect any external modification or conflict, and resolve them.
resume	Resume a server instance that is suspended or in `ADMIN` state.
resync	Resynchronize the configuration files of the specified system component.
resyncAll	Resynchronize the configuration files of all system components.
rolloutApplications	Roll out updates to specified applications on the targeted servers without interrupting service.
rolloutJavaHome	Roll out a new Java Home for targeted servers without interrupting service.
rolloutOracleHome	Roll out a patched Oracle Home to the targeted server without interrupting service.
rollingRestart	Initiate a rolling restart of the targeted servers without interrupting the service.

Table 1-2 (Cont.) WebLogic Server WLST Online Command Summary

This command...	Enables you to...
rolloutUpdate	Roll out updates to targeted servers without interrupting service.
save	Save the edits that have been made but have not yet been saved.
saveDiagnosticImageCaptureFile	Downloads the specified diagnostic image capture.
saveDiagnosticImageCaptureEntryFile	Downloads a specific entry from the diagnostic image capture.
scaleDown	Decrease the number of running dynamic servers in the specified dynamic cluster.
scaleUp	Increase the number of dynamic running servers for the specified dynamic cluster.
serverConfig	Navigate to the last MBean to which you navigated in the configuration MBean hierarchy or to the root of the hierarchy, `DomainMBean`.
serverRuntime	Navigate to the last MBean to which you navigated in the runtime MBean hierarchy or to the root of the hierarchy, `ServerRuntimeMBean`.
set	Set the specified attribute value for the current configuration bean.
setServerGroups	Sets the server groups for the specified server.
setShowLSResult	Specify whether `ls()` should log its output to stdout.
showChanges	Show the changes made by the current user during the current edit session.
showComponentChanges	Show the changes to the configuration of the specified system component on the remote node.
showDebugPatchInfo	Display details about a debug patch on specified targets.
showEditSession	Show information about the specified edit sessions.
showListeners	Show all listeners that are currently defined.
shutdown	Gracefully shut down a running server instance, cluster, or system component.
softRestart	Restart a system component server instance.
start	Start a Managed Server instance, cluster, or system component using Node Manager.
startApplication	Start an application, making it available to users.
startEdit	Start a configuration edit session on behalf of the currently connected user.
startNodeManager	Start Node Manager at default port (5556).
startRecording	Record all user interactions with WLST; useful for capturing commands to replay.
startServer	Start the Administration Server.
state	Returns a map of servers or clusters and their state using Node Manager
stopApplication	Stop an application, making it un available to users.
stopEdit	Stop the current edit session, release the edit lock, and discard unsaved changes.
stopNodeManager	Stop Node Manager.
stopRedirect	Stop the redirection of WLST output to a file.
storeUserConfig	Create a user configuration file and an associated key file.
suspend	Suspend a running server.

Table 1-2 (Cont.) WebLogic Server WLST Online Command Summary

This command...	Enables you to...
threadDump	Display a thread dump for the specified server.
undeploy	Undeploy an application from the specified servers.
undo	Revert all unsaved or unactivated edits.
updateApplication	Update an application configuration using a new deployment plan.
validate	Validate the changes that have been made but have not yet been saved.
viewMBean	Display information about an MBean, such as the attribute names and values, and operations.
writeIniFile	Convert WLST definitions and method declarations to a Python (`.py`) file.
writeTemplate	Writes the domain configuration information to the specified domain template.

WebLogic Server WLST Offline Command Summary

See a summary of the WLST offline commands for Oracle WebLogic Server, listed in alphabetical order.

Table 1-3 WebLogic Server WLST Offline Command Summary

This command...	Enables you to...
addHelpCommand	Adds new command help for a command to an existing command group. Once added to the group, the command (along with a brief description) is displayed in the command list for the group when you enter the `help('commandGroup')` command.
addHelpCommandGroup	Adds a new help command group to those shown by the WLST `help()` command, and specifies the resource bundle in which the help information is defined for the group.
addStartupGroup	Adds a new server startup group.
addTemplate	Extend the current domain using an application or service extension template.
assign	Assign resources to one or more destinations.
cd	Navigate the hierarchy of configuration or runtime beans.
clone	Clones the server object.
closeDomain	Close the current domain.
closestore	Closes a store.
closeTemplate	Close the current domain template.
compactstore	Compacts and defragments the space occupied by a file store.
configToScript	Convert an existing server configuration (`config` directory) to an executable WLST script.
connect	Connect WLST to a WebLogic Server instance.
create	Create a configuration bean of the specified type for the current bean.
createDomain	Creates a WebLogic domain using the specified template.
delete	Delete an instance of a configuration bean of the specified type for the current configuration bean.

Table 1-3 (Cont.) WebLogic Server WLST Offline Command Summary

This command...	Enables you to...
deleteFEHost	Delete the Frontend host for a domain.
deleteStartupGroup	Delete a server startup group.
dumpStack	Display stack trace from the last exception that occurred while performing a WLST action, and reset the stack trace.
dumpstore	Dumps store contents in human-readable format to an XML file.
dumpVariables	Display all variables used by WLST, including their name and value.
exit	Exit WLST from the interactive session and close the scripting shell.
exportDiagnosticData	Execute a query against the specified log file.
exportHarvestedTimeSeriesDataOffline	Export the harvested metric data in offline mode within the specified interval in CSV format.
get	Return the value of the specified attribute.
getDatabaseDefaults	Connect to the database to retrieve schema information.
getFEHostURL	Retrieve the settings for a domain's Frontend Host.
getNodeManagerHome	Get the Node Manager home.
getNodeManagerType	Get the Node Manager type.
getNodeManagerUpgradeOverwriteDefault	Get the value of the Node Manager upgrade overwrite default flag.
getNodeManagerUpgradeType	Get the Node Manager upgrade type used for Node Manager upgrade.
getOldNodeManagerHome	Get the old Node Manager home used for Node Manager upgrade.
getopenstores	Return a list of opened stores (for script access).
getstoreconns	Return a list of connections int he specified store (for script access).
getServerGroups	Retrieve a list of the server groups of which the specified server is a member.
getStartupGroup	Retrieve the server startup group.
getTopologyProfile	Return the domain topology profile for a domain.
listServerGroups	Retrieve a map of the config-groups.xml server groups that are user-expandable.
liststore	List store names, open stores or connections in a store.
loadApplication	Load an application and deployment plan into memory.
loadDB	Load SQL files into a database.
loadProperties	Load property values from a file.
loadTemplates	Load all the selected templates using select template.
ls	List all child beans and/or attributes for the current configuration or runtime bean.
nmConnect	Connect WLST to Node Manager to establish a session.
nmSoftRestart	Restart the specified System Component server instance.
openfilestore	Open a file store.
openjdbcstore	Open a JDBC store.
prompt	Toggle the display of path information at the prompt.
pwd	Display the current location in the configuration or runtime bean hierarchy.

Table 1-3 (Cont.) WebLogic Server WLST Offline Command Summary

This command...	Enables you to...
readDomain	Open an existing WebLogic domain for updating.
readDomainForUpgrade	Open an existing domain for reconfiguration
readTemplate	Open an existing domain template for domain creation.
readTemplateForUpdate	Open an existing domain template for template update.
redirect	Redirect WLST output to the specified filename.
selectCustomTemplate	Select an existing custom domain template or application template for creating a domain
selectTemplate	Select an existing domain template or application template for creating a domain.
set	Set the specified attribute value for the current configuration bean.
setFEHostURL	Set the plain and SSL URLs for a domain's Frontend Host and specifies which is the default.
setOption	Set options related to a WebLogic domain creation or update.
setServerGroups	Set the server groups for the specified server.
setSharedSecretStoreWithPassword	Set the shared secret store and password.
setShowLSResult	Specify whether `ls()` should log its output to stdout.
setStartupGroup	Set the server startup group.
setTopologyProfile	Set the domain topology profile.
showTemplates	Display all currently selected and loaded templates.
showAvailableTemplates	Display all currently available templates for loading.
startNodeManager	Start Node Manager at default port (5556).
startRecording	Record all user interactions with WLST; useful for capturing commands to replay.
startServer	Start the Administration Server.
stopNodeManager	Stop Node Manager.
stopRedirect	Stop the redirection of WLST output to a file.
threadDump	Display a thread dump for the specified server.
unassign	Unassign applications or services from one or more destinations.
unselectCustomTemplate	Deselect the selected custom domain template or application template.
unselectTemplate	Deselect the selected domain template or application template.
updateDomain	Update and save the current domain.
validateConfig	Validate a domain configuration.
writeDomain	Write the domain configuration information to the specified directory.
writeIniFile	Convert WLST definitions and method declarations to a Python (`.py`) file.
writeTemplate	Write the domain configuration information to the specified domain template.

2 WLST Command and Variable Reference

WebLogic Server includes WLST commands that are targeted to specific management and monitoring tasks, such as connecting to a WebLogic Server instance, navigating the hierarchy of configuration or runtime beans, deploying applications, controlling server life cycle, managing diagnostic data, managing JDBC stores, and more.

This chapter includes the following sections:

Overview of WLST Command Categories

Learn about the categories of WLST commands that are available for managing and monitoring Oracle WebLogic Server.

Note:

Refer to Syntax for WLST Commands in *Understanding the WebLogic Scripting Tool* for command syntax requirements.

WLST commands are divided into the following categories.

Table 2-1 WLST Command Categories

Command Category	Description
Browse Commands	Navigate the hierarchy of configuration or runtime beans and control the prompt display.

Table 2-1 (Cont.) WLST Command Categories

Command Category	Description
Control Commands	• Connect to or disconnect from a server. • Create and configure a WebLogic domain or domain template. • Exit WLST.
Customization Commands	Add the command group help and command help that is displayed by the WLST `help()` and `help('`*`commandGroup`*`')` commands.
Deployment Commands	• Deploy, undeploy, and redeploy applications and standalone modules to a WebLogic Server instance. • Update an existing deployment plan. • Interrogate the WebLogic Deployment Manager object. • Start and stop a deployed application.
Diagnostics Commands	Export diagnostic data.
Editing Commands	Interrogate and edit configuration beans.
Information Commands	Interrogate WebLogic domains, servers, and variables, and provide configuration bean, runtime bean, and WLST-related information.
Life Cycle Commands	Manage the life cycle of a server instance or system component instance.
Node Manager Commands	Start, shut down, restart, and monitor WebLogic Server instances and system component instances using Node Manager.
Tree Commands	Navigate among MBean hierarchies.
Store Administration Commands	Manage JDBC stores and file stores.

Browse Commands

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

Table 2-2 Browse Commands for WLST Configuration

Use this command...	To...	Use with WLST...
cd	Navigate the hierarchy of configuration or runtime beans.	Online or Offline
currentTree	Return the current location in the hierarchy.	Online
prompt	Toggle the display of path information at the prompt.	Online or Offline
pwd	Display the current location in the hierarchy.	Online or Offline

cd

Command Category: Browse Commands

Use with WLST: Online or Offline

Description

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

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

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

> **Note:**
>
> The `cmo` variable is initialized to the root of all domain configuration beans when you first connect WLST to a server instance. It reflects the parent configuration bean type until you navigate to an instance. For more information about the `cmo` variable, see Changing the Current Management Object in *Understanding the WebLogic Scripting Tool*.

Syntax

```
cd(mbeanName)
```

Argument	Definition
mbeanName	Path to the bean in the namespace.

Examples

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

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

currentTree

Command Category: Browse Commands

Use with WLST: Online

Description

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

Syntax

```
currentTree()
```

Example

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

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

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

prompt

Command Category: Browse Commands

Use with WLST: Online or Offline

Description

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

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

When you disable the prompt details, to determine your current location in the hierarchy, you can use the `pwd` command, as described in pwd.

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

Syntax

```
prompt(myPrompt)
```

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

Examples

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

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

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

Note:

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

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

pwd

Command Category: Browse Commands

Use with WLST: Online or Offline

Description

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

This command is useful when you have turned off the prompt display of the path information using the `prompt` command, as described in prompt.

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

Syntax

```
pwd()
```

Example

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

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

Control Commands

Use the WLST control commands to perform domain configuration tasks, such as creating or extending a WebLogic domain, configuring server startup groups, rolling out updates to domain components, and more. Table 2-3 lists the control commands you can use to perform the following tasks:

- Connect to or disconnect from a server (`connect` and `disconnect` commands)
- Create a new WebLogic domain from a domain template, similar to the Configuration Wizard (`createDomain`, `readTemplate`, `writeDomain`, and `closeTemplate` commands)
- Update an existing WebLogic domain, offline (`readDomain`, `addTemplate`, `updateDomain`, and `closeDomain` commands)
- Write a domain template (`writeTemplate` command)
- Exit WLST

Table 2-3 Control Commands for WLST Configuration

Use this command...	To...	Use with WLST...
addStartupGroup	Add a new server startup group.	Offline
addTemplate	Extend the current WebLogic domain using an application or service extension template.	Offline
clone	Clone the server object.	Offline
closeDomain	Close the current domain.	Offline
closeTemplate	Close the current domain template.	Offline
connect	Connect WLST to a WebLogic Server instance.	Online or Offline
createDomain	Create a new WebLogic domain using the specified template.	Offline
deleteStartupGroup	Delete a server startup group.	Offline
disconnect	Disconnect WLST from a WebLogic Server instance.	Online
exit	Exit WLST from the interactive session and close the scripting shell.	Online or Offline
getDatabaseDefaults	Connect to the database to retrieve schema information.	Offline

Table 2-3 (Cont.) Control Commands for WLST Configuration

Use this command...	To...	Use with WLST...
getFEHostURL	Retrieve the plain, SSL or default URL for the domain Frontend Host.	Offline
getServerGroups	Retrieve a list of the server groups of which the specified server is a member.	Offline
getStartupGroup	Retrieve the server startup group.	Offline
getTopologyProfile	Return the domain topology profile for a domain.	Offline
listServerGroups	Retrieve a map of the config-groups.xml server groups that are user-expandable.	Offline
loadTemplates	Load all the selected templates using select template.	Offline
readDomain	Open an existing WebLogic domain for updating.	Offline
readDomainForUpgrade	Open an existing domain for reconfiguration.	Offline
readTemplate	Open an existing domain template for domain creation.	Offline
readTemplateForUpdate	Open an existing domain template for template update.	Offline
rolloutApplications	Roll out updates to specified applications on the targeted servers without interrupting service.	Online
rolloutJavaHome	Roll out a new Java Home to the targeted servers without interrupting service.	Online
rolloutOracleHome	Roll out a patched Oracle Home to the targeted server without interrupting service.	Online
rollingRestart	Initiate a rolling restart of the targeted servers without interrupting the service.	Online
rolloutUpdate	Roll out updates to targeted servers without interrupting service.	Online
selectCustomTemplate	Select an existing custom domain template or application template for creating a domain.	Offline
selectTemplate	Select an existing domain template or application template for creating a domain.	Offline
setFEHostURL	Set the plain, SSL and default URLs for the domain Frontend Host.	Offline
setServerGroups	Set the server groups for the specified server.	Offline
setSharedSecretStoreWithPassword	Set the shared secret store and password.	Offline
setStartupGroup	Set the server startup group.	Offline
setToplogyProfile	Set the domain topology profile.	Offline
showAvailableTemplates	Display all currently selected templates for loading.	Offline
showTemplates	Display all currently selected and loaded templates.	Offline
unselectCustomTemplate	Unselect the selected custom domain template or application template.	Offline
unselectTemplate	Unselect a selected domain template or application template.	Offline
updateDomain	Update and save the current domain.	Offline

Table 2-3 (Cont.) Control Commands for WLST Configuration

Use this command...	To...	Use with WLST...
validateConfig	Validate a domain configuration.	Offline
writeDomain	Write the domain configuration information to the specified directory.	Offline
writeTemplate	Writes the configuration information to the specified domain template file.	Online and Offline

addStartupGroup

Command Category: Control Commands

Use with WLST: Offline

Description

Adds a new server startup group based on an existing server group. In the event of an unsupported operation, the command returns a `WLSTException`.

For information about startup groups, see config-groups.xml and startup-plan.xml in *Domain Template Reference*.

Syntax

```
addStartupGroup(server_startup_group_name, server_group_name)
```

Argument	Definition
server_startup_group_name	Name of the new server startup group.
server_group_name	Name of an existing server group within the domain.

Example

The following example creates a startup group called `startup_group_1` from an existing server group called `server_group_1`.

```
addStartupGroup('startup_group_1', 'server_group_1')
```

addTemplate

Command Category: Control Commands

Use with WLST: Offline

Description

Extends the current WebLogic domain using an application or service extension template. Use the Template Builder to create an application or service extension template. See Creating Extension Templates by Using the Domain Template Builder in *Creating Domain Templates Using the Domain Template Builder*.

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

Syntax

```
addTemplate(templateFileName)
```

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

Example

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

```
wls:/offline> readDomain('c:/Oracle/Middleware/user_projects/domains/wlw')
wls:/offline/wlw> addTemplate('c:/Oracle/Middleware/wlserver
/common/templates/wls/DefaultWebApp.jar')
wls:/offline/wlw>
```

Note:

The addTemplate command has been deprecated in this release and will be removed in a future release. Use the selectTemplate and loadTemplates commands instead. See Creating and Updating a WebLogic Domain in *Understanding the WebLogic Scripting Tool*.

closeDomain

Command Category: Control Commands

Use with WLST: Offline

Description

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

Syntax

```
closeDomain()
```

Example

The following example closes the current domain:

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

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

closeTemplate

Command Category: Control Commands

Use with WLST: Offline

Description

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

Syntax

```
closeTemplate()
```

Example

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

```
wls:/offline> readTemplate('c:/Oracle/Middleware/wlserver
/common/templates/wls/wls.jar')
...
wls:/offline/wls> closeTemplate()
wls:/offline>
```

clone

Command Category: Control Commands

Use with WLST: Offline

Description

Clones the original server. In the event of an unsupported operation, the command returns a `WLSTException`.

Syntax

```
clone('originalServerName', 'cloneServerName','type')
```

Argument	Definition
`originalServerName`	The name of the server to clone.
`cloneServerName`	The name of the new cloned server.
`type`	The type of object to clone. Currently, the only supported type is `Server`.

Example

The following example creates a server called `server1_clone` by cloning the `server1` server.

```
wls:/offline/base_domain>clone('server1','server1_clone','Server')
```

connect

Command Category: Control Commands

Use with WLST: Online or Offline

Description

Connects WLST to a WebLogic Server instance.

It requires you to 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 determines the configuration attributes that you can view or modify. (See Default Security Policies for MBeans in *MBean Reference for Oracle WebLogic Server*.)

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.
- 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 user configuration file and key 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.
- Use the credentials that are stored in the Administration Server's `boot.properties` file. By default, when you create an Administration Server in development mode, WebLogic Server encrypts the credentials that were used to create the server and stores them in a `boot.properties` file. When you create an Administration Server in production mode, no `boot.properties` file is created. If your production domain does not contain a `boot.properties` file, you can create one manually; see Creating a Boot Identify File for an Administration Server in *Administering Server Startup and Shutdown for Oracle WebLogic Server*.

  When you run the `connect` command, if there is a `boot.properties` file containing the encrypted username and password for the domain, you do not have to enter the username and password to connect to the Administration Server. You do, however, have to specify the name of the Administration Server in the `connect` command.

Please note:

- If you run the `connect` command in a script without specifying the username and password or user configuration file and key file, a `WSLTException` occurs. In interactive mode, you are prompted for the username and password.
- Oracle strongly recommends that you connect WLST to the server through the SSL port or administration port. If you do not, the following warning message is displayed:

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

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

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

  See Main Steps for Using WLST in Interactive or Script Mode in *Understanding the WebLogic Scripting Tool*.
- If you are connecting to a WebLogic Server instance via HTTP, ensure that the `TunnelingEnabled` attribute is set to `true` for the WebLogic Server instance. See TunnelingEnabled in *MBean Reference for Oracle WebLogic Server*.
- When trying to connect to the WebLogic Server Administration Server from WLST using localhost as the host name, the following message may be displayed if the listen-address attribute of the Administration Server has been restricted to certain IP addresses:

  ```
  javax.naming.CommunicationException [Root exception is
  java.net.ConnectException : <t3://HOST:PORT> : Destination
  unreachable;
  nested exception is: java.net.ConnectException: Connection refused;
  No
  available router to destination
  ```

  You can use either of the following workarounds for this issue:

  – Check that the listen-address attribute of the Administration Server has been set correctly. For example, in the domain configuration file, comment out the <listen-address> line:

  ```
  <server>
     <name>AdminServer</name>
     <ssl>
     .
     .
     .
     </ssl>
     <machine>your_machine</machine>
     <!-- listen-address><your_ip_address></listen-address -->
  </server>
  ```

  – Instead of localhost, use the hostname of the Administration Server in the WLST `connect` command.

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

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

Syntax

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

```
connect([userConfigFile, userKeyFile], [adminServerName], [url], [timeout],
[idd])
connect([url], [adminServerName], [timeout])
```

Argument	Definition
username	Optional. Username of the operator who is connecting WLST to the server. If not specified, WLST processes the command as described above.
password	Optional. Password of the operator who is connecting WLST to the server. If not specified, WLST processes the command as described above.
url	Optional. Listen address and listen port of the server instance, specified using the following format: `[`*`protocol`*`://]`*`listen-address`*`:`*`listen-port`*. If not specified, this argument defaults to `t3://localhost:7001`.
userConfigFile	Optional. Name and location of a user configuration file which contains an encrypted username and password. Use the following syntax for this argument: `userConfigFile='`*`file-system-path`*`'` If not specified, WLST processes the command as described above. When you create a user configuration file, the `storeUserConfig` command uses a key file to encrypt the username and password. Only the key file that encrypts a user configuration file can decrypt the username and password. (See storeUserConfig.)
userKeyFile	Optional. Name and location of the key file that is associated with the specified user configuration file and is used to decrypt it. Use the following syntax for this argument: `userKeyFile='`*`file-system-path`*`'` If not specified, WLST processes the command as described above. See storeUserConfig.
adminServerName	Optional. Name of the Administration Server for the domain. Causes the connect command to use the credentials that are stored in the Administration Server's `boot.properties` file. The `adminServerName` can also be passed with a Managed Server name. Use the following syntax for this argument: `adminServerName='`*`server-name`*`'` This argument is valid only when you start WLST from a domain directory. If the `boot.properties` file for the Administration Server is located in the domain directory, then you do not need to specify this argument. If not specified, WLST processes the command as described above.
timeout	Optional. The number of milliseconds that WLST waits for online commands to complete (return). When you invoke a WLST online command, WLST connects to an MBean Server, invokes an MBean server method, and returns the results of the invocation. If the MBean server method does not return within the timeout period, WLST abandons its invocation attempt. Use the following syntax for this argument: `timeout='`*`milliseconds`*`'` The default value is 0, which indicates that the operation will not time out. Note, however, that a timeout of 5 minutes may occur at a different layer.
idd	Optional. The identity domain for the user. Use the following syntax for this argument, where *`dbUsers`* represents the name of the identity domain: `idd=`*`dbUsers`* This argument defaults to `None`.

Examples

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

```
wls:/offline> connect('adminusername','adminpassword','t3://
localhost:7001')
Connecting to weblogic server instance running at t3://localhost:7001
as
username adminusername...

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

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

wls:/mydomain/serverConfig>
```

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

```
wls:/offline> username = 'adminusername'
wls:/offline> password = 'adminpassword'
wls:/offline> connect(username,password,'t3s://myhost:7001')
Connecting to weblogic server instance running at t3://myhost:7001 as
username adminusername...

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

The following example connects WLST to a WebLogic Server instance using a user configuration and key file to provide user credentials.

```
wls:/offline> connect(userConfigFile='c:/myfiles/
myuserconfigfile.secure',
userKeyFile='c:/myfiles/myuserkeyfile.secure')
Connecting to t3://localhost:7001 with userid username ...

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

The following example shows the prompts that are displayed in interactive mode if you run the command without parameters:

```
wls:/offline> connect()
Please enter your username :username
Please enter your password :
Please enter your server URL [t3://localhost:7001] :
Connecting to t3//localhost:7001 with userid username
```

createDomain

Command Category: Control Commands

Use with WLST: Offline

Description

Creates a WebLogic domain using the specified template.

> **Note:**
>
> The `createDomain` command is similar in functionality to the `unpack` command, as described in The Unpack Command in *Creating Templates and Domains Using the Pack and Unpack Commands*.

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

Syntax

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

Argument	Definition
domainTemplate	Name and location of the domain template from which you want to create a domain.
domainDir	Name of the directory to which you want to write the domain configuration information. Oracle recommends that you create all domains for your environment outside of the Middleware home directory. This makes it easier for you to remove an existing installation or install a new version of WebLogic Server without having to recreate your domains and applications.
user	Name of the default user.
password	Password of the default user.
topologyProfile	Set the topology profile for the domain, either `Compact` or `Expanded`. See Domain Topology Profiles in *Domain Template Reference*.

Example

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

```
wls:/offline> createDomain('c:/Oracle/Middleware/wlserver/common
/templates/wls/wls_medrec.jar','c:/Oracle/Middleware/user_projects/
domains/medrec',
'adminusername','adminpassword')
```

deleteStartupGroup

Command Category: Control Commands

Use with WLST: Offline

Description

Deletes the server startup group. In the event of an unsupported operation, the command returns a `WLSTException`.

Syntax

```
deleteStartupGroup('startup_group_name')
```

Argument	Definition
startup_group_name	Name of the new server startup group.

Example

The following example deletes the server startup group, startup_group_custom.

```
wls:/offline/base_domain> deleteStartupGroup('startup_group_custom')
```

disconnect

Command Category: Control Commands

Use with WLST: Online

Description

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

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

You can connect to another WebLogic Server instance using the `connect` command, as described in connect.

Syntax

```
disconnect(force)
```

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

Example

The following example disconnects from a running server:

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

exit

Command Category: Control Commands

Use with WLST: Online or Offline

Description

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

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

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

Note:

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

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

Syntax

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

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

Example

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

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

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

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

getDatabaseDefaults

Command Category: Control Commands

Use with WLST: Offline

Description

Connects `LocalSvcTblDataSource` to the database server to retrieve schema information and automatically bind these retrieved parameters with local schema components. Data source `LocalSvcTblDataSource` must be configured with the necessary connection parameters before calling `getDatabaseDefaults()`.

Note:

This command can be used only if the domain being created or modified includes Fusion Middleware components that require the use of the Oracle Fusion Middleware Repository Creation Utility (RCU) to load database schemas. The `LocalSvcTblDataSource` connects to the schema corresponding to the ServiceTable component in RCU.

`getDatabaseDefaults()` retrieves the schema password that was set when the schema was first created using RCU. If the schema password was changed after running RCU and before domain creation, you must specify the password again after calling `getDatabaseDefaults()` and before calling `writeDomain()` in a WLST script. For example:

```
getDatabaseDefaults()
cd('/JDBCSystemResource/LocalSvcTblDataSource/JdbcResource/
LocalSvcTblDataSource/JDBCDriverParams/NO_NAME_0')
set('PasswordEncrypted', 'new_password')
writeDomain(DOMAIN_HOME)
```

Syntax

```
getDatabaseDefaults()
```

Example

The following example retrieves schema information for a domain called fmwdomain.

```
wls:/offline/fmwdomain>cd('JDBCSystemResource/LocalSvcTblDataSource/JdbcResource/
LocalSvcTblDataSource')
wls:/offline/fmwdomain>cd('JDBCDriverParams/NO_NAME_0')
wls:/offline/fmwdomain>set('DriverName','oracle.jdbc.OracleDriver')
wls:/offline/fmwdomain>set('URL','jdbc:oracle:thin:@localhost:1522/xe')
wls:/offline/fmwdomain>set('PasswordEncrypted', 'password')
wls:/offline/fmwdomain>cd('Properties/NO_NAME_0')
wls:/offline/fmwdomain>cd('Property/user')
wls:/offline/fmwdomain>cmo.setValue('DEV_STB')
wls:/offline/fmwdomain>getDatabaseDefaults()
```

getFEHostURL

Command Category: Control Commands

Use with WLST: Offline

Description

Retrieves the plain, SSL, or default URL for the domain Frontend Host from the service table.

Syntax

```
getFEHostURL(type)
```

Argument	Definition
type	Specify one of the following types. `default` returns either the plain or SSL URL, depending on which was set as the default when the Frontend Host URL was set. • `plain` • `ssl` • `default`

Example

The following example returns the SSL URL for the Frontend Host in the domain mydomain.

```
wls:/offline> readDomain("/domains/mydomain")
wls:/offline> getFEHostURL("ssl")
wls:/offline> https://myhost.com:7070
```

getServerGroups

Command Category: Control Commands

Use with WLST: Offline

Description

Retrieves a list of the server groups in which the specified server is a member. In the event of an unsupported operation, the command returns a `WLSTException`.

Syntax

```
getServerGroups(server_name)
```

Argument	Definition
server_name	The server name for which associated server groups, if any, will be returned.

Example

The following example gets the server group for the server my_server.

```
wls:/offline/base_domain> getServerGroups('my_server')
'["SERVER-GROUP1-NAME", "SERVER-GROUP2-NAME"]'
```

getStartupGroup

Command Category: Control Commands

Use with WLST: Offline

Description

Retrieves the server startup group associated with a server. In the event of an unsupported operation, the command returns a `WLSTException`.

Syntax

```
getStartupGroups(server_name)
```

Argument	Definition
`server_name`	The server name for which a server startup group, if any, will be returned.

Example

The following example gets the server startup group for the server my_server.

```
wls:/offline/base_domain> getStartupGroup('my_server')
'Startup_Group_1'
```

getTopologyProfile

Returns the domain topology profile for a domain.

Syntax

```
getTopologyProfile()
```

Example

The following example reads the domain `mydomain` and then retrieves the topology profile for the domain (either Compact or Expanded).

```
wls:/offline> readDomain('C:/domains/mydomain')
wls:/mydomain> getTopologyProfile()
'Expanded'
wls:/mydomain>
```

listServerGroups

Command Category: Control Commands

Use with WLST: Offline

Description

Retrieves a map of the config-groups.xml server groups that are user-expandable. In the event of an unsupported operation, the command returns a `WLSTException`.

For information about server groups, see config-groups.xml and startup-plan.xml in *Domain Template Reference*.

Syntax

```
listServerGroups([printout])
```

Argument	Definition
printout	Optional. If `false` (the default) the output will be formatted for programmatic consumption with the server group names returned in a Jython dictionary type that uses the server group name as its key and the server group description as the value.If `true`, the output will be formatted in an easy-to-read table that contains columns for the server group name and that server group's corresponding description.

Example

The following example outputs server groups for programmatic use:

```
wls:/offline> listServerGroups()
'{"SERVER-GROUP1-NAME" : "Server Group 1 Description", "SERVER-GROUP2-
NAME" : "Server Group 2 Description"}'
wls:/offline>
```

The following example outputs server groups for programmatic use:

```
wls:/offline> print listServerGroups('true')
Server Group       | Description
-------------------|--------------------------
SERVER-GROUP1-NAME | Server Group 1 Description
SERVER-GROUP2-NAME | Server Group 2 Description
wls:/offline>
```

readDomain

Command Category: Control Commands

Use with WLST: Offline

Description

Opens an existing WebLogic domain for updating.

WLST offline provides read and write access to the configuration data that is persisted in the `config` directory for the WebLogic domain, 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.

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

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

See Navigating and Interrogating MBeans in *Understanding the WebLogic Scripting Tool*.

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

Syntax

```
readDomain(domainDirName)
```

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

Example

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

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

readDomainForUpgrade

Command Category: Control Commands

Use with WLST: Offline

Description

The `readDomainForUpgrade()` function performs three basic tasks:

- Upgrades the core WebLogic Server configuration to the new version
- Reads the resulting domain into memory similarly to the existing readDomain function
- Selects the required reconfiguration templates and adds them to the domain similarly to the existing addTemplate function.

Note that if the core WebLogic Server upgrade portion of this is successful, it cannot be undone. The addition of the reconfiguration templates must be committed via the updateDomain function later in the session in order to take effect.

Syntax

```
readDomainForUpgrade(domain_dir), [properties]
```

Argument	Definition
domain_dir	The path of the domain directory that you want to open for reconfiguration.
properties	Optional. WebLogic Server defined properties for core WebLogic Server infrastructure upgrade. This may be a `java.util.Properties` object, a Python Directory object, or a semicolon delimited string of `name=value` pairs.

Example

The following example opens the domain located at C:\domains\medrec for upgrade:

```
>wls:/offline> readDomainForUpgrade('c:/domains/medrec')
```

readTemplate

Command Category: Control Commands

Use with WLST: Offline

Description

Opens an existing domain template for domain creation and optionally specifies the config-groups topology to use (full WebLogic Server only).

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

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

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

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

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

Syntax

```
readTemplate(templateFileName), [topologyprofile]
```

Argument	Definition
templateFileName	Name of the JAR file corresponding to the domain template.
topologyprofile	Optional. The name of the config-groups topology profile to use (`Compact` or `Expanded`). This argument can be used only in a full WebLogic Server installation and applies only if the template defines these topologies. If omitted, the `Expanded` topology is used. **Note:** This argument does not apply in standalone system component (for example, OHS) installations, in which a restricted WLST tool is provided.

Example

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

```
wls:/offline> readTemplate('c:/Oracle/Middleware/wlserver/common/templates
/wls/wls_medrec.jar')
wls:/offline/wls_medrec>
```

> **Note:**
>
> The `readTemplate` command has been deprecated in this release and will be removed in a future release. Use the `selectTemplate` and `loadTemplates` commands instead. See Creating and Updating a WebLogic Domain in *Understanding the WebLogic Scripting Tool*.

setFEHostURL

Command Category: Control Commands

Use with WLST: Offline

Description

Sets the plain and SSL URLs for the domain Frontend Host and specifies whether the plain or SSL URL is the default.

Syntax

```
setFEHostURL(plain, ssl, isDefaultPlain)
```

Argument	Definition
plain	Sets the plain URL for the Frontend Host in the following format: `http://`*`host`*`:`*`port`*
ssl	Sets the SSL URL for the Frontend Host in the following format: `https://`*`host`*`:`*`port`* **Note:** If you are using the plain URL for the frontend host, you must still specify a default SSL URL. If you have only set up a plain connection to the Frontend Host, you can use the same URL for both plain and SSL. Otherwise, they must be different.
isPlainDefault	Boolean. If set to `true`, the plain URL is the default for the Frontend Host. If set to `false`, the SSL URL is hte default for the Frontend Host.

Example

The following example sets the plain and SSL URLs for the Frontend Host of the domain mydomain, and specifies that the default is the SSL URL.

```
wls:/offline> readDomain("/domains/mydomain")
wls:/offline> setFEHostURL("http://www.myhost.com:7733","http://
www.myhost.com:7733"4455",
false)
```

setServerGroups

Command Category: Control Commands

Use with WLST: Online and Offline

Description

Sets the user-expandable server groups to which you want the specified server to belong. After executing this function, the server belongs only to the specified server groups.

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

Syntax

```
setServerGroups(serverName, serverGroups, [timeout], [skipEdit])
```

Argument	Definition
serverName	The server name that you want to associate with the specified user-configurable server groups.
serverGroups	A Python list of the user-configurable server groups to associate with the specified server.
timeout	Optional. Applies only when using `setServerGroups` in online mode. Specifies the amount of time the connection can be inactive before the connection times out. The default is 120000 milliseconds.
skipEdit	Optional. Applies only when using `setServerGroups` in online mode. If set to `true`, the `setServerGroups` online operation does not start and activate a new WLST `edit()` session and instead relies on an existing `edit()` session. The default is `true`.

Example

The following example associates the server my_server with user-expandable server groups `server_group_1` and `server_group_2`.

```
wls:/offline/base_domain> groups="server_group_1, server_group_2"
wls:/offline/base_domain> setServerGroups('my_server', groups)
```

setStartupGroup

Command Category: Control Commands

Use with WLST: Offline

Description

Sets the server startup group for a server. In the event of an unsupported operation, the command returns a `WLSTException`.

Syntax

```
setStartupGroup(server_name, server_startup_group_name)
```

Argument	Definition
server_name	The name of the server for which you want to set the startup group.
server_startup_group_name	The name of an existing server startup group within the domain.

Example

The following example sets the startup group for the server `my_server` to `startup_group_1`.

```
wls:/offline/base_domain> setStartupGroup('my_server', 'startup_group_1')
```

setSharedSecretStoreWithPassword

Command Category: Control Commands

Use with WLST: Offline

Description

Sets the shared secret store and password for a shared database in a domain which includes Fusion Middleware products. In the event of an unsupported operation, the command returns a `WLSTException`.

Syntax

```
setSharedSecretStoreWithPassword(sharedSecretStore, secretStorePassword)
```

Argument	Definition
sharedSecretStore	The name of the shared secret store.
secretStorePassword	The password for the shared secret store.

Example

The following example sets the password for secret store `store1` to `password` for the domain `base_domain`.

```
wls:/offline/base_domain>setSharedSecretStoreWithPassword(store1, password)
```

updateDomain

Command Category: Control Commands

Use with WLST: Offline

Description

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

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

Syntax

```
updateDomain()
```

Example

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

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

writeDomain

Command Category: Control Commands

Use with WLST: Offline

Description

Writes the domain configuration information to the specified directory.

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

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

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

Note:

The name of the WebLogic domain is derived from the name of the domain directory. For example, for a domain saved to `c:/Oracle/Middleware/user_projects/domains/myMedrec`, the domain name is `myMedrec`.

When updating an existing domain, you must use updateDomain in place of `writeDomain()`.

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

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

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

Syntax

```
writeDomain(domainDir)
```

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

Example

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

```
wls:/offline> readTemplate('c:/Oracle/Middleware/wlserver/common/templates
/wls/wls.jar')
...
wls:/offline/base_domain> writeDomain('c:/Oracle/Middleware/user_projects/domains/
base_domain')
```

writeTemplate

Command Category: Control Commands

Use with WLST: Online and Offline

Description

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

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

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

Note:

The `writeTemplate` command is similar in functionality to the `pack` command. See The Pack Command in *Creating Templates and Domains Using the pack and unpack Commands*.

Syntax

```
writeTemplate(templateName, [timeout])
```

Argument	Definition
templateName	Name of the domain template to store the domain configuration information.
timeout	Optional. Applies only when using `writeTemplate` in online mode. Specifies the amount of time the connection to the remote server can be inactive before the connection times out.

Example

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

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

validateConfig

Command Category: Control Commands

Use with WLST: Online or Offline

Description

Validates a domain configuration using the supplied option. In the event of a validation error, the command returns a `WLSTException`.

Syntax

```
validateConfig(option)
```

Argument	Definition
option	Specify one of the following options: **InternalPortConflict**—Validate Administration Server and Managed Server listen port conflicts in the current domain configuration. **ExternalPortConflict**—Validate Administration Server and Managed Server listen port conflicts against ports in use by active processes on the current machine (host). **KeyStorePasswords**—Checks whether any selected template that contains a valid metadata does not yet have the credential store created for it. **ClusterFrontEnd**—Validates whether the format of the configured front-end host and front-end http (or https) values. It also validates whether the front-end http (or https) value is missing when front-end host is set, or whether the front-end host value is missing when the front-end http (or https) value is set.

Example

The following example validates the internal listen port configuration in the domain `base_domain`:

```
wlst:/offline/base_domain>validateConfig('InternalPortConflict')
```

setTopologyProfile

Command Category: Control Commands

Use with WLST: Offline

Description

Sets the domain topology profile `(Compact or Expanded)` to use for that domain. This should be set before selecting the domain template. If the topology profile is not set, then the default topology is used.

Syntax

```
setTopologyProfile(Topology Profile)
```

Argument	Definition
`Topology Profile`	The name of the config-groups topology profile to use `(Compact or Expanded)`. If the topology profile is not set, then the Expanded topology is used.

Example

The following example sets the domain topology profile as Compact for a domain:

```
wlst:/offline> setTopologyProfile('Compact')
```

selectTemplate

Command Category: Control Commands

Use with WLST: Offline

Description

Selects an existing domain template or application template for creating a domain.

Syntax

```
selectTemplate(Template Name,Template Version)
```

Argument	Definition
`Template Name`	Name of the domain template as defined in the template descriptor file.
`Template Version`	Optional. Version of the domain template. If there are multiple templates with the same name, then the template version must be specified. Not specifying the template version will raise an error if there are multiple templates with the same name in the Middleware Home.

Example

The following example selects the specified domain template for creating a domain:

```
wls:/offline> selectTemplate('Basic WebLogic Server Domain','12.2.1.0')
```

rolloutApplications

Command Category: Control Commands

Use with WLST: Online

Description

Rolls out updates to specified applications deployed on the targeted servers without interrupting service. The target can be a domain, a cluster, or a comma-separated list of servers.

This operation involves graceful shutdown of the servers, updating the applications, restarting the Node Manager, and starting the servers again.

Syntax

```
rolloutApplications(target, applicationPropertiesFile, [options])
```

Argument	Definition
`target`	Name of the domain, cluster, or a comma-separated list of servers on which the rollout will take effect.
`applicationPropertiesFile`	The location of the text file containing the properties of each application that will be updated.
`options`	Optional. Comma-separated list of options, specified as name-value pairs. Valid options include: • **isDryRun**—Boolean value. If set to `true`, the operation will be evaluated but not executed. This option defaults to `false`. • **autoRevertOnFailure**—Boolean value. If set to `true`, the operation automatically reverts on failure. If set to `false`, the operation stops on failure and waits for the user to resume the operation. This options defaults to `true`. • **isSessionCompatible**—Boolean value. Set to `true` if the sessions between the patched and unpatched versions of OracleHome are compatible. This affects the session handling time and graceful server shutdown time. Set to `false` if special consideration must be taken to preserve unpatched sessions. This could impact the time it takes for the rollout to complete. This option defaults to `false`. • **migrationProperties**—The full path to a JSON file that defines singleton service migrations to be performed during the rollout. See **Preparing to Migrate Singleton Services** in *Administering Zero Downtime Patching Workflows*. • **shutdownTimeout**—Time (in seconds) WLST waits for a server to gracefully shutdown before shutting it down forcefully. If the patched sessions are not compatible with the unpatched sessions, then forceful shutdown of the servers on the last node may result in loss of session data. A value of less than 1 will be ignored. • **extension**—The full path to the location of the extension jar file, optionally followed by a comma-separated list of script parameters specified as name-value pairs. If you specify the script parameters using this option, then these parameter values will override the values specified in the JSON file. • **extensionProperties**—The full path to the `extensionProperties.json` file that is used to specify one or more extension jars.

Example

The following example shows how to use the `rolloutApplications` command to roll out the applications defined in the JSON-formatted application properties file `/u01/scratch/app_update.json` to all servers in `Cluster1`.

```
wls:/myDomain/serverConfig/> progress=rolloutApplications('Cluster1', '/u01/scratch/
app_update.json')
```

rolloutJavaHome

Command Category: Control Commands

Use with WLST: Online

Description

Updates the targeted servers to use a new JavaHome without interrupting the service.

This operation results in graceful shutdown of the servers, the location of Java Home being updated, the Node Manager being restarted, and the servers being restarted without interrupting the service for the user.

Syntax

```
rolloutJavaHome(target, javaHomeDirectory, [options])
```

Argument	Definition
`target`	Name of the domain, cluster, or a comma-separated list of servers on which the rollout will take effect.
`javaHomeDirectory`	The location of the new Java Home to use. This must refer to a valid Java Home path installed on each machine.

Argument	Definition
options	Optional. A comma-separated list of options, specified as name-value pairs. Valid options include: • **isDryRun**—Boolean value. If set to `true`, the operation is evaluated but not executed. This option defaults to `false`. • **autoRevertOnFailure**—Boolean value. If set to `true`, the operation automatically reverts on failure. If set to `false`, the operation stops on failure and waits for the user to resume the operation. This options defaults to `true`. • **isSessionCompatible**—Boolean value. Set to `true` if the sessions between the patched and unpatched versions of Oracle Home are compatible. This affects the session handling time and graceful server shutdown time. Set to `false` if special consideration must be taken to preserve unpatched sessions. This could impact the time it takes for the rollout to complete. This option defaults to `false`. • **migrationProperties**—The full path to a JSON file that defines singleton service migrations to be performed during the rollout. See Preparing to Migrate Singleton Services in *Administering Zero Downtime Patching Workflows*. • **shutdownTimeout**—Time (in seconds) WLST waits for a server to gracefully shutdown before shutting it down forcefully. If the patched sessions are not compatible with the unpatched sessions, then forceful shutdown of the servers on the last node may result in loss of session data. A value of less than 1 is ignored. • **extension**—The full path to the location of the extension jar file, optionally followed by a comma-separated list of script parameters specified as name-value pairs. If you specify the script parameters using this option, then these parameter values will override the values specified in the JSON file. • **extensionProperties**—The full path to the `extensionProperties.json` file that is used to specify one or more extension jars.

Example

The following example shows how to use the `rolloutJavaHome` command to roll out a new Java Home to the cluster `Cluster1`. The new Java Home location is `/u01/jdk1.8.0_50`.

```
wls:/myDomain/serverConfig/> progress=rolloutJavaHome('Cluster1', '/u01/
jdk1.8.0_50')
```

rolloutOracleHome

Command Category: Control Commands

Use with WLST: Online

Description

Rolls out a patched Oracle Home to the targeted servers or reverts your targeted servers to use the previous unpatched Oracle Home.

This operation results in graceful shutdown of the servers, the Node Manager being restarted, and the servers being restarted without interrupting the service for the user.

Syntax

```
rolloutOracleHome(target, rolloutOracleHome, backupOracleHome, isRollback, [options])
```

Argument	Definition
target	Name of the domain, cluster, or a comma-separated list of servers on which the rollout will take effect.
rolloutOracleHome	The location of the archive or local directory containing the version of Oracle Home to roll out, that will replace the existing Oracle Home.
backupOracleHome	The path of the local directory to which the existing Oracle Home will be moved.
isRollback	Boolean value. Allows the user to specify that the change being rolled out to a domain is to a previous patch release of Oracle Home. This information is important in determining whether the Administration Server should be updated first or last. Set to `True` if the Oracle Home being rolled out has an older patch version than the current Oracle Home.
options	Optional. Comma-separated list of options, specified as name-value pairs. Valid options include: • **isDryRun**—Boolean value. If set to `true`, the operation is evaluated but not executed. This option defaults to `false`. • **autoRevertOnFailure**—Boolean value. If set to `true`, the operation automatically reverts on failure. If set to `false`, the operation stops on failure and waits for the user to resume the operation. This options defaults to `true`. • **isSessionCompatible**—Boolean value. Set to `true` if the sessions between the patched and unpatched versions of Oracle Home are compatible. This affects the session handling time and graceful server shutdown time. Set to `false` if special consideration must be taken to preserve unpatched sessions. This could impact the time it takes for the rollout to complete. This option defaults to `false`. • **migrationProperties**—The full path to a JSON file that defines singleton service migrations to be performed during the rollout. See **Preparing to Migrate Singleton Services** in *Administering Zero Downtime Patching Workflows*. • **shutdownTimeout**—Time (in seconds) WLST waits for a server to gracefully shutdown before shutting it down forcefully. If the patched sessions are not compatible with the unpatched sessions, then forceful shutdown of the servers on the last node may result in loss of session data. A value of less than 1 is ignored. • **extension**—The full path to the location of the extension jar file, optionally followed by a comma-separated list of script parameters specified as name-value pairs. If you specify the script parameters using this option, then these parameter values will override the values specified in the JSON file. • **extensionProperties**—The full path to the `extensionProperties.json` file that is used to specify one or more extension jars.

Example

The following example shows how to use the `rolloutOracleHome` command to roll out a new Oracle Home to the domain myDomain. The JAR file for the patched Oracle Home is located at `/net/wls/wls_patched.jar`. The original Oracle Home will be moved to `/u01/Oracle_Home_backup`. The process will not automatically revert if it fails.

```
wls:/myDomain/serverConfig/> progress=rolloutOracleHome('myDomain', '/net/wls/
wls_patched.jar', '/u01/Oracle_Home_backup', autoRevertOnFailure=FALSE)
)
```

rollingRestart

Command Category: Control Commands

Use with WLST: Online

Description

Initiates a rolling restart of all servers in a domain or all servers in a specific cluster or clusters without interrupting the service. This command provides the ability to sequentially restart servers.

This operation involves the graceful shutdown of the servers, and the servers being restarted without interrupting the service for the user.

Syntax

```
rollingRestart(target, [options])
```

Argument	Definition
target	Name of the domain, cluster, or a comma-separated list of servers on which the rollout takes effect.

Argument	Definition
options	Optional. Comma-separated list of options, specified as name-value pairs. Valid options include: • **isDryRun**—Boolean value. If set to `true`, the operation is evaluated but not executed. This option defaults to `false`. • **autoRevertOnFailure**—Boolean value. If set to `true`, the operation automatically reverts on failure. If set to `false`, the operation stops on failure and waits for the user to resume the operation. This options defaults to `true`. • **isSessionCompatible**—Boolean value. Set to `true` if the sessions between the patched and unpatched versions of Oracle Home are compatible. This affects the session handling time and graceful server shutdown time. Set to `false` if special consideration must be taken to preserve unpatched sessions. This could impact the time it takes for the rollout to complete. This option defaults to `false`. • **migrationProperties**—The full path to a JSON file that defines singleton service migrations to be performed during the rollout. See **Preparing to Migrate Singleton Services** in *Administering Zero Downtime Patching Workflows*. • **shutdownTimeout**—Time (in seconds) WLST waits for a server to gracefully shutdown before shutting it down forcefully. If the patched sessions are not compatible with the unpatched sessions, then forceful shutdown of the servers on the last node may result in loss of session data. A value of less than 1 is ignored. • **extension**—The full path to the location of the extension jar file, optionally followed by a comma-separated list of script parameters specified as name-value pairs. If you specify the script parameters using this option, then these parameter values will override the values specified in the JSON file. • **extensionProperties**—The full path to the `extensionProperties.json` file that is used to specify one or more extension jars.

Example

The following example uses the `rollingRestart` command to restart all servers in the domain `myDomain`, one at a time. It then, uses the returned progress object to show the progress and state of the operation.

```
wls:/myDomain/serverConfig/> progress = rollingRestart('myDomain')
 wls:/myDomain/serverConfig/> progress.getProgressString()
    '[MgmtOrchestration:2192004]Workflow wf9 Running: 43 / 80'
 wls:/myDomain/serverConfig/> progress.getStatus()     'SUCCESS'
```

rolloutUpdate

Command Category: Control Commands

Use with WLST: Online

Description

Updates the targeted servers to use Oracle Home, Java Home, or applications deployed on the server without interrupting the service or causing loss of session. This command allows for any combination of those updates, depending on the optional parameters that are specified. If the user doesn't specify the Oracle Home, Java Home, or application properties parameters, then `rollingRestart` is executed.

Syntax

```
rolloutUpdate(target, [rolloutOracleHome, backupOracleHome, isRollback],
[javaHomeDirectory], [applicationPropertiesFile], [options])
```

Argument	Definition
target	Name of the domain, cluster, or a comma-separated list of servers on which the rollout takes effect.
rolloutOracleHome	The location of the archive or local directory containing the version of Oracle Home to roll out, and that replaces the existing Oracle Home. See rolloutOracleHome.
backupOracleHome	The path of the local directory to which the existing Oracle Home is moved. See rolloutOracleHome.
javaHomeDirectory	This argument must be specified if you want to update Java Home. See rolloutOracleHome.
applicationPropertiesFile	This argument must be specified if you want to update applications deployed on the server. See rolloutApplications.
isRollback	This argument must be specified if you want to update Oracle Home. See rolloutOracleHome.

Argument	Definition
options	Optional. Comma-separated list of options, specified as name-value pairs. Valid options include: • **isDryRun**—Boolean value. If set to `true`, the operation is evaluated but not executed. This option defaults to `false`. • **autoRevertOnFailure**—Boolean value. If set to `true`, the operation automatically reverts on failure. If set to `false`, the operation stops on failure and waits for the user to resume the operation. This options defaults to `true`. • **isSessionCompatible**—Boolean value. Set to `true` if the sessions between the patched and unpatched versions of Oracle Home are compatible. This affects the session handling time and graceful server shutdown time. Set to `false` if special consideration must be taken to preserve unpatched sessions. This could impact the time it takes for the rollout to complete. This option defaults to `false`. • **migrationProperties**—The full path to a JSON file that defines singleton service migrations to be performed during the rollout. See **Preparing to Migrate Singleton Services** in *Administering Zero Downtime Patching Workflows*. • **shutdownTimeout**—Time (in seconds) WLST waits for a server to gracefully shutdown before shutting it down forcefully. If the patched sessions are not compatible with the unpatched sessions, then forceful shutdown of the servers on the last node may result in loss of session data. A value of less than 1 is ignored. • **extension**—The full path to the location of the extension jar file, optionally followed by a comma-separated list of script parameters specified as name-value pairs. If you specify the script parameters using this option, then these parameter values will override the values specified in the JSON file. • **extensionProperties**—The full path to the `extensionProperties.json` file that is used to specify one or more extension jars.

Example

The following example uses the `rolloutUpdate` command to roll out a new Oracle Home and a new Java Home to the Administration Server. The JAR file for the patched Oracle Home is located at `/net/wls/wls_patched.jar`. The original Oracle Home will be moved to `/u01/Oracle_Home_backup`. The new Java Home location is `/u01/jdk1.8.0_50`.

```
wls:/myDomain/serverConfig/> progress=rolloutUpdate('AdminServer', '/net/wls/
wls_patched.jar', '/u01/Oracle_Home_backup', '/u01/jdk1.8.0_50')
```

selectCustomTemplate

Command Category: Control Commands

Use with WLST: Offline

Description

Select an existing custom domain template or application template for creating a domain.

Syntax

```
selectCustomTemplate(Template Path)
```

Argument	Definition
`Template Path`	Name of the JAR file corresponding to the domain template.

Example

The following example selects the `wls.jar` custom domain template for creating a domain:

```
wls:/offline> selectCustomTemplate('C:/Oracle/Middleware/WLS/wlserver/common/
templates/wls/wls.jar')
```

unselectTemplate

Command Category: Control Commands

Use with WLST: Offline

Description

Deselect the selected domain template or application template during the process of domain creation.

Syntax

```
unselectTemplate(Template Name, Template Version)
```

Argument	Definition
`Template Name`	Name of the domain template as defined in the template descriptor file.
`Template Version`	Optional. Version of the domain template to be deselected. If there are multiple templates with the same name, then the template version must be specified.

Example

The following example deselects the specified domain template during the process of domain creation:

```
wls:/offline> unselectTemplate('Basic WebLogic Server Domain','12.2.1.0')
```

unselectCustomTemplate

Command Category: Control Commands

Use with WLST: Offline

Description

Deselect the selected custom domain template or application template during the process of domain creation.

Syntax

```
unselectCustomTemplate(Template Path)
```

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

Example

The following example deselects the `wls.jar` custom domain template during the process of domain creation:

```
wls:/offline> unselectCustomTemplate('C:/Oracle/Middleware/WLS/wlserver/common/
templates/wls/wls.jar')
```

loadTemplates

Command Category: Control Commands

Use with WLST: Offline

Description

Loads all the selected templates using the `selectTemplate` command.

Syntax

```
loadTemplates()
```

Example

The following example loads the selected template.

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

readTemplateForUpdate

Command Category: Control Commands

Use with WLST: Offline

Description

Opens an existing domain template for template update. If you are updating the domain template, you must use readTemplateForUpdate instead of readTemplate.

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

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

WebLogic Server configuration beans exist within hierarchical structures. In the WLST file system, configuration bean hierarchies correspond to drives; types and instances are directories; attributes and operations are files. WLST traverses the hierarchical structure of configuration beans using commands such as cd, ls, and pwd in a similar way that you would navigate a file system in a UNIX or Windows command shell. After navigating to a configuration bean instance, you interact with the bean using WLST commands. See Navigating and Interrogating MBeans in *Understanding the WebLogic Scripting Tool*.

Note:

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

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

Syntax

```
readTemplateForUpdate(templateFileName)
```

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

Example

The following example opens the `wls.jar` domain template for template update:

```
wls:/offline> readTemplateForUpdate('C:/Oracle/Middleware/WLS/wlserver/common/
templates/wls/wls.jar')
wls:/offline/base_domain>
```

showAvailableTemplates

Command Category: Control Commands

Use with WLST: Offline

Description

Displays all currently available templates for loading.

Syntax

```
showAvailableTemplates('showHidden', 'verbose', 'includeApplied')
```

Argument	Definition
showHidden	Optional. Displays hidden templates. The default value is `false`.
verbose	Optional. Displays the complete template location path. The default value is `false`.
includeApplied	Optional. Displays applied templates. The default value is `false`.

Example

```
wls:/offline> showAvailableTemplates('false','true','false')
```

showTemplates

Command Category: Control Commands

Use with WLST: Offline

Description

Displays all currently selected and loaded templates.

Syntax

```
showTemplates()
```

Example

The following example displays all currently selected and loaded templates:

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

Customization Commands

Use the WLST customization commands to add the command group help and command help that is listed by the WLST `help()` and `help('`*`commandGroup`*`')` commands. Table 2-4 lists and summarizes these commands. See Adding Integrated Help for Custom Commands in *Understanding the WebLogic Scripting Tool*.

Table 2-4 Customization Commands for WLST Configuration

This command...	Enables you to...	Use with WLST...
addHelpCommandGroup	Adds a new help command group to those shown by the WLST `help()` command.	Online or Offline
addHelpCommand	Adds new command help for a command to an existing command group. Once added to the group, the command (along with a brief description) is displayed in the command list for the group when you enter the `help('commandGroup')` command.	Online or Offline

addHelpCommandGroup

Command Category: Customization Commands

Use with WLST: Online or Offline

Description

Adds a new command help group to those shown by the WLST `help()` command, and specifies the resource bundle in which the help information is defined for the group.

Syntax

```
addHelpCommandGroup(commandGroup, resourceBundleName)
```

Argument	Definition
commandGroup	Use a unique name for the command group. Do not use a command group name that is already shown by the WLST `help()` command.
resourceBundleName	Represents either a class name or property resource file name. The resource bundle contains help text for entries for the command group using a standard pattern. The resource bundle name will be passed to `ResourceBundle.getBundle(...)`. Multiple command groups can use the same resource bundle. The resource bundle must be present in the classpath. See Adding Integrated Help for Custom Commands in *Understanding the WebLogic Scripting Tool* for information on how to define the help text for each command group and command. For more information on resourceBundles and localization, refer to `http://docs.oracle.com/javase/8/docs/api/java/util/ResourceBundle.html`.

Examples

The following example adds the `boot` command group to the list of groups shown by the `help()` command, and specifies that the help text is located in the property resource file 'myhelp':

```
wls:/offline> addHelpCommandGroup('boot','myhelp')
```

The following example adds the `boot` command group to the list of groups shown by the `help()` command, and specifies that the help text is located in the class `foo.bar.MyResourceBundleClass`:

```
wls:/offline> addHelpCommandGroup('boot','foo.bar.MyResourceBundleClass')
```

addHelpCommand

Command Category: Customization Commands

Use with WLST: Online or Offline

Description

Adds new command help for a command to an existing command group. Once added to the group, the command (along with a brief description) is displayed in the command list for the group when you enter the `help('commandGroup')` command. You can also specify whether or not the command is listed by the `help('online')` and `help('offline')` commands.

Syntax

```
addHelpCommand(commandName,commandGroup,[offline=false, online=false])
```

Argument	Definition
commandName	The name of the command as defined in the command group specified by commandGroup.
commandGroup	The commandGroup to which the command belongs.

Argument	Definition
`online`	Optional. Boolean value that determines whether or not the command shows up in the `help('online')` output. The default value is `'false'`.
`offline`	Optional. Boolean value that determines whether or not the command shows up in the `help('offline')` output. The default value is `'false'`.

Example

The following example shows how to add the online command `bootDB` to the listing output by the `help('boot')` and `help('online')` commands:

```
wls:/offline> addHelpCommand('bootDB','boot',online='true',offline='false')
```

Deployment Commands

Use the WLST deployment commands to perform tasks such as deploying and undeploying applications, updating a deployment plan, interrogating the WebLogic Deployment Manager object, starting and stopping applications, and more.
Table 2-5 lists and summarizes these commands.
See Understanding WebLogic Server Deployment in *Deploying Applications to Oracle WebLogic Server*.

Table 2-5 Deployment Commands for WLST Configuration

This command...	Enables you to...	Use with WLST...
deploy	Deploy an application to a WebLogic Server instance.	Online
distributeApplication	Copy the deployment bundle to the specified targets.	Online
getWLDM	Return the WebLogic `DeploymentManager` object.	Online
listApplications	List all applications that are currently deployed in the WebLogic domain.	Online
loadApplication	Load an application and deployment plan into memory.	Online and Offline
redeploy	Redeploy a previously deployed application.	Online
startApplication	Start an application, making it available to users.	Online
stopApplication	Stop an application, making it unavailable to users.	Online
undeploy	Undeploy an application from the specified servers.	Online
updateApplication	Update an application configuration using a new deployment plan.	Online
appendToExtensionLoader	Distributes code source jar to targets and uses it to extend the WebLogic Extension Loader.	Online

deploy

Command Category: Deployment Commands

Use with WLST: Online

Description

Deploys an application to a WebLogic Server instance.

The `deploy` command returns a `WLSTProgress` object that you can access to check the status of the command. See WLSTProgress Object in *Understanding the WebLogic Scripting Tool*. In the event of an error, the command returns a `WLSTException`.

Note:

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

Syntax

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

Argument	Definition
`appName`	Name of the application or standalone Java EE module to be deployed.
`path`	Name of the application directory, archive file, or root of the exploded archive directory to be deployed.
`targets`	Optional. Comma-separated list of the targets. Each target may be qualified with a Java EE module name (for example, *`module1@server1`*) enabling you to deploy different modules of the application archive on different servers. This argument defaults to the server to which WLST is currently connected.
`stageMode`	Optional. Staging mode for the application you are deploying. Valid values are `stage`, `nostage`, and `external_stage`. See Controlling Deployment File Copying with Staging Modes in *Deploying Applications to Oracle WebLogic Server*. If you do not specify a stage mode, the default stage mode is used. On the Administration Server, the default stage mode is `nostage` and on Managed Servers, it is `stage`.
`planPath`	Optional. Name of the deployment plan file. The filename can be absolute or relative to the application directory. This argument defaults to the `plan/plan.xml` file in the application directory, if one exists.

Argument	Definition
options	Optional. Comma-separated list of deployment options, specified as name-value pairs. Valid options include: • **adminMode**—Boolean value specifying whether to start the Web application with restricted access. This option defaults to `false`. • **altDD**—Location of the alternate application deployment descriptor on the Administration Server. • **altWlsDD**—Location of the alternate WebLogic application deployment descriptor on the Administration Server. • **archiveVersion**—Archive version number. • **block**—Boolean value specifying whether WLST should block user interaction until the command completes. This option defaults to `true`. If set to `false`, WLST returns control to the user after issuing the command; you can query the `WLSTProgress` object to determine the status of the command. If you are importing WLST as a Jython module, as described in Importing WLST as a Jython Module in *Understanding the WebLogic Scripting Tool*, *block* is always set to `true`. • **clusterDeploymentTimeout**—Time, in milliseconds, granted for a cluster deployment task on this application. • **createPlan**—Boolean value indicating that user would like to create a default plan. This option defaults to `false`. • **defaultSubmoduleTargets**—Boolean value indicating that targeting for qualifying JMS submodules should be derived by the system, see Using Sub-Module Targeting with JMS Application Modules in *Deploying Applications to Oracle WebLogic Server*. Default value is true. • **deploymentPrincipalName**—String value specifying the principal for deploying the file or archive during server starts (static deployment; it does not affect the current deployment task). Make sure the user exists. This option adds `<deployment-principal-name>` to the `<app-deployment>` element in the `config.xml` file. • **deploymentOrder**—Integer. Specify the deployment order of the application. The application with the lowest deployment order value is deployed first. • **forceUndeployTimeout**—Time (in seconds) to wait for in-flight work to complete before undeploying the application. • **gracefulIgnoreSessions**—Boolean value specifying whether the graceful production to admin mode operation should ignore pending HTTP sessions. This option defaults to `false` and only applies if `gracefulProductionToAdmin` is set to `true`. • **gracefulProductionToAdmin**—Boolean value specifying whether the production to Admin mode operation should be graceful. This option defaults to `false`. • **libImplVersion**—Implementation version of the library, if it is not present in the manifest. • **libraryModule**—Boolean value specifying whether the module is a library module. This option defaults to `false`.

Argument	Definition
`options` *(Continued)*	• **libSpecVersion**—Specification version of the library, if it is not present in the manifest. • **planStageMode**—Staging mode for the deployment plan. Valid values are stage, nostage, or external_stage. • **planVersion**—Plan version number. • **remote**—Boolean value specifying whether the operation will be remote from the file system that contains the source. Use this option when you are on a different machine from the Administration Server and the deployment files are already at the specified location where the Administration Server is located. This option defaults to false. • **retireGracefully**—Retirement policy to gracefully retire an application only after it has completed all in-flight work. This policy is only meaningful for stop and redeploy operations and is mutually exclusive to the retire timeout policy. • **retireTimeout**—Time (in seconds) WLST waits before retiring an application that has been replaced with a newer version. This option default to `-1`, which specifies graceful timeout. • **rmiGracePeriod**—Time (in seconds) WLST waits for RMI requests before retiring an application that has been replaced with a newer version. RMI requests arriving within a grace period of prior request scheduled within the grace period will be accepted, and otherwise rejected. This option default to -1, which specifies no grace period. • **securityModel**—Security model. Valid values include: `DDOnly`, `CustomRoles`, `CustomRolesAndPolicies`, and `Advanced`. • **securityValidationEnabled**—Boolean value specifying whether security validation is enabled. • **subModuleTargets**—Submodule level targets for JMS modules. For example, `submod@mod-jms.xml@target | submoduleName@target`. • **timeout**—Time (in milliseconds) that WLST waits for the deployment process to complete before canceling the operation. A value of 0 indicates that the operation will not time out. This argument defaults to 300,000 ms (or 5 minutes). • **upload**—Boolean value specifying whether the application files are uploaded to the WebLogic Server Administration Server's upload directory prior to deployment. Use this option when the Administration Server cannot access the application files through the file system. This option defaults to false. • **versionIdentifier**—Version identifier.

Examples

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

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

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

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

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

See WLSTProgress Object in *Understanding the WebLogic Scripting Tool*.

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

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

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

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

The following example shows how to set the application version (`appVersion`) to a unique identifier to support production (side-by-side) redeployment. This example deploys the `demoApp` application in the archive file located at `c:/myapps/demos/app/demoApp.ear`, and sets the application and archive version numbers to the specified values.

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

See Redeploying Applications in a Production Environment in *Deploying Applications to Oracle WebLogic Server*.

The following example shows how to deploy `myapp.ear` in `nostage` mode using the deployment plan `plan.xml`.

```
wls:/mydomain/serverConfig/Servers> progress=deploy('myApp',
'c:/myapps/myapp.ear', 'mywar@webserver,myjar@ejbserver','
'c:/myapps/plan.xml')
...Deployment of 'myApp' is successful
```

distributeApplication

Command Category: Deployment Commands

Use with WLST: Online

Description

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

The `distributeApplication` command returns a `WLSTProgress` object that you can access to check the status of the command. See WLSTProgress Object in *Understanding the WebLogic Scripting Tool*. In the event of an error, the command returns a `WLSTException`.

Syntax

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

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

Example

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

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

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

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

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

See WLSTProgress Object in *Understanding the WebLogic Scripting Tool*.

appendToExtensionLoader

Command Category: Deployment Commands

Use with WLST: Online

Description

Distributes code source jar to targets and use it to extend the WebLogic Extension Loader.

Syntax

```
appendToExtensionLoader([options])
```

Argument	Definition
options	This command supports the following options: **targets**—A comma-separated list of server names, cluster names, or both. The default value is `All`. **source**—Specifies the code source jar to distribute. **upload**—Transfers the specified code source jar to the Administration Server before distribution. Use this option when you are on a remote machine and you cannot copy the code source jar to the Administration Server by other means. The code source jar is uploaded to the WebLogic Administration Server's upload directory before the distribution.

Example

The following example distributes a code source jar to the managed servers in the `Cluster-1` and adds the jars to the WebLogic Extension Loader's class search space for the running servers:

```
appendToExtensionLoader(source='custom.jar', targets='Cluster-1')
```

getWLDM

Command Category: Deployment Commands

Use with WLST: Online

Description

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

Syntax

```
getWLDM()
```

Example

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

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

listApplications

Command Category: Deployment Commands

Use with WLST: Online

Description

Lists all applications that are currently deployed in the WebLogic domain.

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

Syntax

```
listApplications()
```

Example

The following example lists all the applications currently deployed in `mydomain`.

```
wls:/mydomain/serverConfig> listApplications()
SamplesSearchWebApp
asyncServletEar
jspSimpleTagEar
ejb30
webservicesJwsSimpleEar
ejb20BeanMgedEar
xmlBeanEar
extServletAnnotationsEar
examplesWebApp
apache_xbean.jar
mainWebApp
jdbcRowSetsEar
```

loadApplication

Command Category: Deployment Commands

Use with WLST: Online and Offline

Description

Loads an application and deployment plan into memory. When used in online mode, you can connect only to the Administration Server; you cannot connect to a Managed Server.

The `loadApplication` command returns a `WLSTPlan` object that you can access to make changes to the deployment plan. See WLSTPlan Object in *Understanding the WebLogic Scripting Tool*. In the event of an error, the command returns a `WLSTException`.

Note:

The `loadApplication` command does not support loading an application from a remote machine. The application must be on the same machine where the Administration Server is running.

Syntax

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

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

Example

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

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

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

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

See WLSTPlan Object in *Understanding the WebLogic Scripting Tool*.

redeploy

Command Category: Deployment Commands

Use with WLST: Online

Description

Reloads classes and redeploys a previously deployed application.

The `redeploy` command returns a `WLSTProgress` object that you can access to check the status of the command. See WLSTProgress Object in *Understanding the WebLogic Scripting Tool*.

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

See Overview of Common Deployment Scenarios in *Deploying Applications to Oracle WebLogic Server*.

Syntax

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

Argument	Definition
`appName`	Name of the application to be redeployed.
`planPath`	Optional. Name of the deployment plan file. The filename can be absolute or relative to the application directory. This argument defaults to the `plan/plan.xml` file in the application directory, if one exists.
`options`	Optional. Comma-separated list of deployment options, specified as name-value pairs. For a list of valid deployment options, see *`options`* argument description in deploy. In addition, the following deployment option can be specified for the `redeploy` command: • **appPath**—Name of the archive file or root of the exploded archive directory to be redeployed. • **delta**—A list of file paths or names relative to the root of the staged application that are to be replaced. • **deploymentPrincipalName**—String value specifying the principal for redeploying the file or archive during server starts. You can use this option to overwrite the current `<deployment-principal-name>` in the `config.xml` file.

Example

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

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

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

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

See WLSTProgress Object in *Understanding the WebLogic Scripting Tool*.

startApplication

Command Category: Deployment Commands

Use with WLST: Online

Description

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

The `startApplication` command returns a `WLSTProgress` object that you can access to check the status of the command. See WLSTProgress Object in *Understanding the WebLogic Scripting Tool*. In the event of an error, the command returns a `WLSTException`.

Syntax

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

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

Example

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

```
wls:/mydomain/serverConfig/Servers> progress=startApplication('BigApp',
stageMode='NOSTAGE', adminMode='false')
Starting the application...
Successfully started the application.
```

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

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

See WLSTProgress Object in *Understanding the WebLogic Scripting Tool*.

stopApplication

Command Category: Deployment Commands

Use with WLST: Online

Description

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

The `stopApplication` command returns a `WLSTProgress` object that you can access to check the status of the command. See WLSTProgress Object in *Understanding the WebLogic Scripting Tool*.

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

Syntax

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

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

Argument	Definition
options	Optional. Comma-separated list of deployment options, specified as name-value pairs. For a list of valid deployment options, see *options* argument description in deploy.

Example

The following example stops the `BigApp` application.

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

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

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

See WLSTProgress Object in *Understanding the WebLogic Scripting Tool*.

undeploy

Command Category: Deployment Commands

Use with WLST: Online

Description

Undeploys an application from the specified servers.

The `undeploy` command returns a `WLSTProgress` object that you can access to check the status of the command. See WLSTProgress Object in *Understanding the WebLogic Scripting Tool*. In the event of an error, the command returns a `WLSTException`.

See Overview of Common Deployment Scenarios in *Deploying Applications to Oracle WebLogic Server*.

Syntax

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

Argument	Definition
appName	Deployment name for the deployed application.
targets	Optional. List of the target servers from which the application will be removed. If not specified, defaults to all current targets.

Example

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

```
wls:/mydomain/serverConfig> undeploy('businessApp', timeout=60000)
Undeploying application businessApp ...
<Jul 20, 2005 9:34:15 AM EDT> <Info> <J2EE Deployment SPI> <BEA-260121>
<Initiating undeploy operation for application, businessApp [archive: null],
to AdminServer.>
Completed the undeployment of Application with status
Current Status of your Deployment:
Deployment command type: undeploy
Deployment State       : completed
Deployment Message     : no message
wls:/mydomain/serverConfig>
```

updateApplication

Command Category: Deployment Commands

Use with WLST: Online

Description

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

The `updateApplication` command returns a `WLSTProgress` object that you can access to check the status of the command. See WLSTProgress Object in *Understanding the WebLogic Scripting Tool*. In the event of an error, the command returns a `WLSTException`.

Syntax

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

Argument	Definition
appName	Name of the application, as specified in the current `plan.xml` file.
planPath	Optional. Name of the new deployment plan file. The filename can be absolute or relative to the application directory.
options	Optional. Comma-separated list of deployment options, specified as name-value pairs. For a list of valid deployment options, see *options* argument description in deploy.

Example

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

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

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

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

See WLSTProgress Object in *Understanding the WebLogic Scripting Tool*.

Diagnostics Commands

Use the WLST diagnostics commands to retrieve diagnostics data by executing queries against the WebLogic Diagnostics Framework (WLDF) data stores. Table 2-6 lists and summarizes these commands. For more information about WLDF, see What Is the WebLogic Diagnostics Framework? in *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*.

Table 2-6 Diagnostic Command for WLST Configuration

This command...	Enables you to...	Use with WLST...
activateDebugPatch	Activate a debug patch on the specified targets.	Online
captureAndSaveDiagnosticImage	Capture a diagnostic image and downloads it to the client.	Online
createSystemResourceControl	Create a diagnostics system resource control using specified descriptor file.	Online
deactivateAllDebugPatches	Deactivate all debug patches on the specified targets.	Online
deactivateDebugPatches	Deactivate debug patches on the specified targets.	Online
destroySystemResourceControl	Destroy a diagnostics system resource control.	Online
disableSystemResource	Deactivate a diagnostic system resource control that is activated on a server instance.	Online
dumpDiagnosticData	Poll for live data matching the Harvester configuration for a particular WLDF system resource and dump it to a local file.	Online
enableSystemResource	Activate a diagnostic system resource on a server instance	Online
exportDiagnosticData	Execute a query against the specified log file.	Offline
exportDiagnosticDataFromServer	Execute a query on the server side and retrieves the exported WebLogic Diagnostic Framework (WLDF) data.	Online
exportHarvestedTimeSeriesData	Export the harvested metric data within the specified interval in CSV format.	Online
exportHarvestedTimeSeriesDataOffline	Export the harvested metric data in offline mode within the specified interval, in CSV format.	Offline
getAvailableCapturedImages	Return a list of the previously captured diagnostic images.	Online
getAvailableDiagnosticDataAccessorNames	Get the diagnostic data accessor names currently available on a server.	Online
listDebugPatches	List active and available debug patches on the specified targets.	Online

Table 2-6 (Cont.) Diagnostic Command for WLST Configuration

This command...	Enables you to...	Use with WLST...
listDebugPatchTasks	List debug patch tasks from the specified targets.	Online
listSystemResourceControls	List the diagnostic system resources that are currently deployed on a server instance.	Online
mergeDiagnosticData	Merges a set of data files.	Online
purgeCapturedImages	Purge the diagnostic image files on the server as per the specified age criteria.	Online
purgeDebugPatchTask	Purge debug patch tasks from the specified targets.	Online
showDebugPatchInfo	Display details about a debug patch on the specified targets.	Online
saveDiagnosticImageCaptureFile	Download the specified diagnostic image capture.	Online
saveDiagnosticImageCaptureEntryFile	Download a specific entry from the diagnostic image capture.	Online

activateDebugPatch

Command Category: Diagnostics Commands

Use with WLST: Online

Description

Activates a debug patch on the specified targets. While connected to the Administration Server, the optional target parameter can be specified to activate the debug patch on multiple Managed Servers and clusters. If the target parameter is not specified, the debug patch will be activated only on the connected server.

The `activateDebugPatch` command returns an array of tasks, each element corresponding to the activation activity on an affected target server instance.

Syntax

```
activateDebugPatch(patch, [options])
```

Argument	Description
patch	Name of the debug patch to be activated. The patch must exist at the applicable target servers. Use the `listDebugPatches` command to view the list of available and active patches on the target servers.

Argument	Description
options	**app**—Activates the debug patch within the scope of the specified application. If not specified, the debug patch will be activated at the system level. The default value is `None`. **module**—Activates the debug patch within the scope of the specified module in the application. This option is ignored if the `app` option is set to `None` and the debug patch is activated at the system level. **target**—Administration Server only. A comma-separated list of server names, cluster names, or both where the debug patch will be activated. The default value is `None`.

Example

The following example activates the `dyndebug_app01.jar` debug patch within the scope of the application, `myapp` on the cluster, `myCluster`.

```
wls:/dyndebugDomain/serverConfig> tasks=activateDebugPatch('dyndebug_app01.jar',
app='myapp', target='myCluster')
```

captureAndSaveDiagnosticImage

Command Category: Diagnostics Commands

Use with WLST: Online

Descriptions

Captures a diagnostic image and download it to the client.

Syntax

```
captureAndSaveDiagnosticImage([options])
```

Argument	Description
options	This command supports the following options: • **Server**—Administration Server only. The name of the target server. The default value is `None`. This option has been replaced by the `Target` option; all WLST scripts should be updated to use `Target` instead. • **Target**—Administration Server only. A comma-separated list of server names, cluster names, or both. The default value is `None`. • **outputFile**—The file in which to save the image. • **outputDir**—The destination directory into which image files will be copied if multiple files are retrieved. The default value is the current directory.

Example

The following example captures a diagnostic image on the server, `myserver`, and members of the cluster, `Cluster-0`, and retrieves it on the client:

```
wls:/mydomain/serverConfig>
captureAndSaveDiagnosticImage(Target='myserver,Cluster-0')
Capture and save diagnostics images
```

```
Image created on the server diagnostic_image_MS1_2013_09_19_16_20_49.zip
Saving diagnostic image diagnostic_image_MS1_2013_09_19_16_20_49.zip from server MS1
to diagnostic_image_MS1_2013_09_19_16_20_49.zip
Image created on the server diagnostic_image_MS2_2013_09_19_16_20_51.zip
Saving diagnostic image diagnostic_image_MS2_2013_09_19_16_20_51.zip from server MS2
to diagnostic_image_MS2_2013_09_19_16_20_51.zip
Image created on the server diagnostic_image_myserver_2013_09_19_16_20_53.zip
Saving diagnostic image diagnostic_image_myserver_2013_09_19_16_20_53.zip from server
myserver to diagnostic_image_myserver_2013_09_19_16_20_53.zip
```

createSystemResourceControl

Command Category: Diagnostics Commands

Use with WLST: Online

Description

Creates a diagnostics resource from a diagnostic descriptor file without changing the domain configuration. Note that the resource remains in memory only until the server is shut down; the resource is not deployed the next time the server is booted. That is, this command effects a run-time change only, not a configuration change.

Syntax

```
createSystemResourceControl(resourceName, descriptorFile, [options])
```

Argument	Definition
resourceName	Name of the diagnostic resource. **Note:** The resource must be enabled separately through the `enableSystemResource()` function, and the features in the resource itself must be enabled.
descriptorFile	Local path to the diagnostic descriptor file.
options	**Server**—Administration Server only. The target server name. The default value is `None`. This option has been replaced by the `Target` option; all WLST scripts should be updated to use `Target` instead. **Target**—A comma-separated list of server names, cluster names, or both. The default value is `None`. **enabled**—If set to `true`, the system resource is enabled after it has been created. The default value is `false`. Note that the resource can be enabled or disabled separately by using the `enableSystemResource` and `disableSystemResource` commands. Note also that the features in the resource must be enabled in order to be active.

Examples

The following example creates a system resource control called `myExternalResource` from the diagnostic descriptor file `mywldf.xml` with a server target `myserver` and a cluster target `mycluster`. The resource control is not enabled.

```
wls:/mydomain/serverRuntime>createSystemResourceControl('myExternalResource',
'C:/temp/mywldf.xml', Target='myserver,mycluster')
```

The following example creates and enables a system resource control called `myExternalResource` from the diagnostic descriptor file `mywldf.xml` with a server target `myserver` and a cluster target `mycluster`.

```
wls:/mydomain/serverRuntime>createSystemResourceControl('myExternalResource',
'C:/temp/mywldf.xml', Target='myserver,mycluster', enabled='true')
```

The following example creates and enables a system resource control called `myExternalResource` from the diagnostic descriptor file `mywldf.xml`. Because no targets are specified, the target defaults to the currently connected server.

```
wls:/mydomain/serverRuntime>createSystemResourceControl('myExternalResource',
'C:/temp/mywldf.xml', enabled='true')
```

deactivateAllDebugPatches

Command Category: Diagnostics Commands

Use with WLST: Online

Description

Deactivates all debug patches on the specified targets. While connected to the Administration Server, the optional `target` parameter can be specified to deactivate all debug patches on multiple Managed Servers and clusters. If the `target` parameter is not specified, the debug patches only on the connected server will be deactivated.

The `deactivateAllDebugPatches` command returns an array of tasks, each element corresponding to the deactivation activity on an affected target server instance.

Syntax

```
deactivateAllDebugPatches([target])
```

Argument	Description
`target`	Optional. Administration Server only. A comma-separated list of server names, cluster names, or both on which the debug patches will be deactivated. The default value is `None`. If not specified, the debug patches only on the connected server will be deactivated.

Example

The following example deactivates all debug patches on the cluster `myCluster`.

```
wls:/dyndebugDomain/serverConfig>
tasks=deactivateAllDebugPatches(target='myCluster')
```

deactivateDebugPatches

Command Category: Diagnostics Commands

Use with WLST: Online

Description

Deactivates a debug patch on the specified targets. While connected to the Administration Server, the optional `target` parameter can be specified to deactivate the debug patch on multiple Managed Servers and clusters. If the `target` parameter is not specified, the debug patch only on the connected server will be deactivated.

The `deactivateDebugPatches` command returns an array of tasks, each element corresponding to the deactivation activity on an affected target server instance.

Syntax

```
deactivateDebugPatches(patches, [options])
```

Argument	Description
patches	Comma-separated list of debug patches to be deactivated.
options	**app**—Deactivates debug patches within the scope of the specified application. If not specified, the debug patch will be deactivated at the system level. The default value is `None`. **module**—Deactivates debug patches within the scope of the specified module in the application. This option is ignored if the `app` option is set to `None` and the debug patch is deactivated at the system level. **target**—Administration Server only. A comma-separated list of server names, cluster names, or both on which the debug patches will be deactivated. The default value is `None`.

Example

The following example deactivates a list of debug patches within the scope of an application myapp on the cluster `myCluster`.

```
wls:/dyndebugDomain/serverConfig>
tasks=deactivateDebugPatches('dyndebug_app01.jar,dyndebug_app02.jar', app='myapp',
target='myCluster')
```

destroySystemResourceControl

Command Category: Diagnostics Commands

Use with WLST: Online

Description

Destroys a diagnostics resource that was deployed from an external descriptor using the `createSystemResourceControl()` function.

Note:

To disable a diagnostics resource without destroying it, use the `disableSystemResource()` command.

Syntax

```
destroySystemResourceControl(resourceName, [options])
```

Argument	Definition
resourceName	Name of the diagnostic resource. **Note:** The resource must be enabled separately through the `enableSystemResource()` function, and the features in the resource itself must be enabled.
options	**Server**—Administration Server only. The name of the server on which the resource is currently deployed. The default value is `None`. This option has been replaced by the `Target` option; all WLST scripts should be updated to use `Target` instead. **Target**—A comma-separated list of server names, cluster names, or both on which the resource is currently deployed. The default value is `None`.

Example

The following example destroys a diagnostics system resource control called `myExternalResource` that was deployed on the `myserver` server.

```
wls:/mydomain/serverRuntime>destroySystemResourceControl('myExternalResource',
Target='myserver')
```

disableSystemResource

Command Category: Diagnostic Commands

Use with WLST: Online

Descriptions

Deactivates a diagnostic system resource that is enabled on a server instance.

Syntax

```
disableSystemResource(resourceName, [options])
```

Argument	Definition
resourceName	Name of the diagnostic resource. **Note:** The resource must be enabled separately through the `enableSystemResource()` function, and the features in the resource itself must be enabled.
options	**Server**—Administration Server only. The target server name. The default value is `None`. This option has been replaced by the `Target` option; all WLST scripts should be updated to use `Target` instead. **Target**—A comma-separated list of server names, cluster names, or both. The default value is `None`.

Example

The following example deactivates a diagnostics system resource called `mySystemResource` that is activated on the `myserver` server.

```
wls:/mydomain/serverRuntime>disableSystemResource('mySystemResource',Target='myserver')
```

dumpDiagnosticData

Command Category: Diagnostics Commands

Use with WLST: Online

Description

Polls for live diagnostics data matching the Harvester configuration for a particular WLDF system resource at the specified frequency and duration, and dumps it to a local file. The data is written in CSV format.

If WLDF detects a change in the set of data collected between successive sampling periods, a new output file is created and an informational message is displayed. The creation of multiple output files can result if the sampling interval is particularly long. You can merge these data files using the mergeDiagnosticData command.

Syntax

```
dumpDiagnosticData(resourceName, filename, frequency, duration, [options])
```

Argument	Definition
resourceName	The name of the system resource from which the data is polled. **Note:** The resource must be enabled separately through the `enableSystemResource()` function, and the features in the resource itself must be enabled.
filename	The name of the file into which the data is to be dumped.
frequency	The frequency (in milliseconds) at which data is polled from the system resource Harvester.
duration	The total duration (in milliseconds) for which the data will be collected and saved.
options	**Server**—Administration Server only. The name of the server on which the system resource is running. **dateFormat**—A SimpleDateFormat pattern. The default is "EEE MM/dd/YY k:mm:ss:SSS z".

Example

The following example dumps diagnostic Harvester data for `mySystemResource` on the server `myserver` to the local file C:\temp\temp.dat, and sets the date format to HH:mm:ss:SSS.

```
wls:/mydomain/serverRuntime>dumpDiagnosticData('mySystemResource',
'C:/temp/temp.dat',1000,6000,Server='myserver',dateFormat='HH:mm:ss:SSS')

Connecting to http://localhost:7001 with userid weblogic ...
Dumping data for system resource mySystemResource in file temp.data every second  for
6 seconds.
```

enableSystemResource

Command Category: Diagnostics Commands

Use with WLST: Offline

Description

Activates a diagnostic system resource on a server instance.

Syntax

```
enableSystemResource(resourceName, [options]
```

Argument	Definition
resourceName	Name of the diagnostic system resource.
options	**Server**—Administration Server only. The target server name. The default value is `None`. This option has been replaced by the `Target` option; all WLST scripts should be updated to use `Target` instead. **Target**—A comma-separated list of server names, cluster names, or both. The default value is `None`.

Example

The following example activates the diagnostic system resource `mySystemResource` on the `myServer` server.

```
wls:/mydomain/
serverRuntime>enableSystemResource('mySystemResource',Target='myserver')
```

exportDiagnosticData

Command Category: Diagnostics Commands

Use with WLST: Offline

Description

Executes a query against the specified log file. The results are saved to an XML, CSV or TXT file. The default output format is XML.

See Accessing Diagnostic Data With the Data Accessor in *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*.

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

Syntax

```
exportDiagnosticData([options])
```

Argument	Description
options	Optional. Comma-separated list of export diagnostic options, specified as name-value pairs. Valid options include: • **beginTimestamp**—Timestamp (inclusive) of the earliest record to be added to the result set. This option defaults to 0. • **endTimestamp**—Timestamp (exclusive) of the latest record to be added to the result set. This option defaults to `Long.MAX_VALUE`. • **exportFileName**—Name of the file to which the data is exported. This option defaults to `export.xml`. • **logicalName**—Logical name of the log file being read. Valid values include: `HarvestedDataArchive`, `EventsDataArchive`, `ServerLog`, `DomainLog`, `HTTPAccessLog`, `WebAppLog`, `ConnectorLog`, and `JMSMessageLog`. This option defaults to `ServerLog`. • **logName**—Base log filename containing the log data to be exported. This option defaults to `myserver.log` in the current directory. • **logRotationDir**—Directory containing the rotated log files. This option defaults to "`.`" (which is the same directory in which the log file is stored). • **query**—Expression specifying the filter condition for the data records to be included in the result set. This option defaults to "" (empty string), which returns all data. See WLDF Query Language in *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*. • **storeDir**—Location of the diagnostic store for the server. This option defaults to `../data/store/diagnostics`. • **format**—The format in which the data will be exported. The supported formats are `xml`, `txt` and `csv`. The default format is `xml`. • **last**—The timestamp range specification for the last n records. When specified, the `beginTimestamp` and `endTimestamp` options are ignored. The format is *XX*`d` *YY*`h` *ZZ*`m`, for example, `1d 5h 30m` specifies data that is one day, five hours and 30 minutes old. You can specify any combination of day, hour, and minute components in any order.

Example

The following example executes a query against the `ServerLog` named `myserver.log` and stores the results in the file named `myExport.xml`.

```
wls:/offline/mydomain>exportDiagnosticData(logicalName='ServerLog',
logName='myserver.log', exportFileName='myExport.xml')
Input parameters: {'elfFields': '', 'logName': 'myserver.log', 'logRotationDir':
'.', 'endTimestamp': 9223372036854775807L, 'exportFileName': 'export.xml',
'storeDir': '../data/store/diagnostics', 'logicalName': 'ServerLog',
'query': '', 'beginTimestamp': 0}

Exporting diagnostic data to export.xml
<Aug 2, 2005 6:58:21 PM EDT> <Info> <Store> <BEA-280050> <Persistent store
 "WLS_DIAGNOSTICS" opened: directory="c:\Oracle\Middleware
\wlserver\server\data\store\diagnostics"
 writePolicy="Disabled" blockSize=512 directIO=false driver="wlfileio2">
```

exportDiagnosticDataFromServer

Command Category: Diagnostics Commands

Use with WLST: Online

Description

Executes a query on the server side and retrieves the exported WebLogic Diagnostic Framework (WLDF) data. The results are saved to an XML, CSV or TXT file. The default output format is XML. This command requires that you have a secure connection to the Managed Server.

See Accessing Diagnostic Data With the Data Accessor in *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*.

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

Syntax

```
exportDiagnosticDataFromServer([options])
```

Argument	Definition
`options`	Optional. Comma-separated list of export diagnostic options, specified as name-value pairs. Valid options include: • **beginTimestamp**—Timestamp (inclusive) of the earliest record to be added to the result set. This option defaults to 0. • **endTimestamp**—Timestamp (exclusive) of the latest record to be added to the result set. This option defaults to `Long.MAX_VALUE`. • **exportFileName**—Name of the file to which the data is exported. This option defaults to `export.xml`. • **logicalName**—Logical name of the log file being read. Valid values include: `HarvestedDataArchive`, `EventsDataArchive`, `ServerLog`, `DomainLog`, `HTTPAccessLog`, `WebAppLog`, `ConnectorLog`, and `JMSMessageLog`. This option defaults to `ServerLog`. • **query**—Expression specifying the filter condition for the data records to be included in the result set. This option defaults to "" (empty string), which returns all data. • **server**—The name of the Managed Server from which to retrieve the data. This option can be used only if you are currently connected to the Administration Server. By default, data is retrieved only from the server to which WLST is connected. • **format**— The format in which the data will be exported. The supported formats are `xml`, `txt` and `csv`. The default format is `xml`. • **last**—The timestamp range specification for the last n records. When specified, the `beginTimestamp` and `endTimestamp` options are ignored. The format is *`XX`*`d` *`YY`*`h` *`ZZ`*`m`, for example, `1d 5h 30m` specifies data that is one day, five hours and 30 minutes old. You can specify any combination of day, hour, and minute components in any order.

Example

The following example executes a query against the `HTTPAccessLog` and stores the results in the file named `myExport.xml`.

```
wls:/mydomain/serverRuntime>
exportDiagnosticDataFromServer(logicalName="HTTPAccessLog",
exportFileName="myExport.xml")
```

exportHarvestedTimeSeriesData

Command Category: Diagnostics Commands

Use with WLST: Online

Description

Exports harvested metric data from the diagnostic archive for a particular server-scoped WLDF system resource.

See Configuring the Harvester for Metric Collection in *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*.

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

Syntax

```
exportHarvestedTimeSeriesData(wldfSystemResource, [options])
```

Argument	Definition
wldfSystemResource	The name of the WLDF system resource from which the harvested data is exported.
options	Comma-separated list of export diagnostic options, specified as name-value pairs. Valid options include: • **beginTimestamp**— Beginning timestamp of the harvested data to be retrieved. • **endTimestamp**—Ending timestamp of the harvested data to be retrieved. • **exportFile**— The path to the file in which the exported data is written in the CSV format. • **server**—The name of the server on which the WLDF system resource is running, and the server from which the harvested data will be retrieved. This option can be used only if WLST is connected to the Administration Server. If not specified, the data will be retrieved from the server to which WLST is connected. • **last**—The timestamp range specification for the last n records. When specified, the beginTimestamp and endTimestamp options are ignored. The format is `XXd YYh ZZm`. For example, `1d 5h 30m` specifies data that is one day, five hours and 30 minutes old. You can specify any combination of day, hour, and minute components in any order. • **dateFormat**—A pattern string conforming to the java.text.SimpleDateFormat syntax in which the timestamp for the harvested data is rendered. The format is `EEE MM/dd/YY k:mm:ss:SSS z`.

Example

The following example exports the harvested diagnostic data from the WLDF system resource, `wldf-1` running on the `myserver` server, and writes the data to the file named `export.csv`.

```
wls:/mydomain/serverRuntime> exportHarvestedTimeSeriesData('wldf-1')
Getting diagnostic data from Server myserver.
Opening new capture file export.csv...
```

exportHarvestedTimeSeriesDataOffline

Command Category: Diagnostics Commands

Use with WLST: Offline

Description

Exports the harvested metric data from the diagnostic archive for a WLDF system resource in the offline mode when the server is not running.

See Configuring the Harvester for Metric Collection in *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*.

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

Syntax

```
exportHarvestedTimeSeriesDataOffline(wldfSystemResource, [options])
```

Argument	Definition
`wldfSystemResource`	The name of the WLDF system resource from which the harvested data is exported.
`options`	Optional. Comma-separated list of export diagnostic options, specified as name-value pairs. Valid options include: • **beginTimestamp**— Beginning timestamp of the harvested data to be retrieved. • **endTimestamp**—Ending timestamp of the harvested data to be retrieved. • **exportFile**— The path to the file in which the exported data is written in the CSV format. • **last**—The timestamp range specification for the last n records. When specified, the beginTimestamp and endTimestamp options are ignored. The format is *`XX`*`d` *`YY`*`h` *`ZZ`*`m`. For example, `1d 5h 30m` specifies data that is one day, five hours and 30 minutes old. You can specify any combination of day, hour, and minute components in any order. • **dateFormat**—A `java.text.SimpleDateFormat` pattern string in which the timestamp for harvested data is rendered. The format is `EEE MM/dd/YY k:mm:ss:SSS z`. • **storeDir**— The path to the WLDF store directory. The diagnostic data store for a server instance is located in the `data/store/diagnostics` directory under the server's root directory.

Example

The following example exports the harvested diagnostic data from the WLDF system resource, `wldf-1` in offline mode, and writes the data to the file named `export.csv`.

```
wls:/offline> exportHarvestedTimeSeriesDataOffline(wldfSystemResource='wldf-1')
Opening new capture file export.csv ...
```

getAvailableCapturedImages

Command Category: Diagnostics Commands

Use with WLST: Online

Description

Returns, as an array of strings, a list of the previously captured diagnostic images that are stored in the image destination directory configured on the server. The default directory is `SERVER\logs\diagnostic_images`.

This command is useful for identifying a diagnostic image capture that you want to download, or for identifying a diagnostic image capture from which you want to download a specific entry.

See Configuring and Capturing Diagnostic Images in *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*.

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

Syntax

```
getAvailableCapturedImages([options])
```

Argument	Definition
options	**server**—(Administration Server only) The server from which to obtain the list of available images. The default value is `None`.

Examples

The following example returns an array of strings named `images`, which contains a list of the diagnostic image capture files available in the image destination directory, and prints the entries contained in the diagnostic image named `diagnostic_image_myserver_2009_06_15_14_58_36.zip`.

```
wls:/mydomain/serverRuntime>images=getAvailableCapturedImages()
Connecting to http://localhost:7001 with userid weblogic ...
wls:/mydomain/serverRuntime>print images]
'diagnostic_image_myserver_2009_06_15_14_58_36.zip']
]
```

The following example returns an array of strings named `images`, which contains a list of the diagnostic image capture files available in the image destination directory, for the Managed Server `MS1` and prints the entries contained in the diagnostic images named `diagnostic_image_MS1_2013_09_18_15_59_31.zip` and `diagnostic_image_MS1_2013_09_18_16_02_58.zip`.

```
wls:/mydomain/serverRuntime>images=getAvailableCapturedImages(Server='MS1')
Connecting to http://localhost:7001 with userid weblogic ...
wls:/mydomain/serverRuntime>print images
['diagnostic_image_MS1_2013_09_18_15_59_31.zip',
'diagnostic_image_MS1_2013_09_18_16_02_58.zip']
```

getAvailableDiagnosticDataAccessorNames

Command Category: Diagnostics Commands

Use with WLST: Online

Description

Gets the logical names of diagnostic data accessor instances currently available on a server, and returns them as an array of string values.

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

Syntax

```
getAvailableDiagnosticDataAccessorNames([options])
```

Argument	Definition
options	**server**— The server from which the list of available data accessor names will be retrieved. If not specified, the data will be retrieved from the current server to which WLST is connected. This option is applicable only when WLST is connected to the Administration Server. The default value is `None`.

Example

The following example retrieves the diagnostic data accessor names from `myserver` server and returns them as an array of string values.

```
wls:/mydomain/serverRuntime>names=getAvailableDiagnosticDataAccessorNames()
Getting diagnostic data from Server myserver.
wls:/mydomain/serverRuntime>print names
array(java.lang.String,['HarvestedDataArchive', 'EventsDataArchive',
'DataSourceLog', 'DomainLog', 'HTTPAccessLog', 'ServerLog'])
```

listDebugPatches

Command Category: Diagnostics Commands

Use with WLST: Online

Description

Lists the active and available debug patches on the specified targets. While connected to the Administration Server, the optional target parameter can be specified to list debug patches on multiple Managed Servers and clusters.

Syntax

```
listDebugPatches([target])
```

Argument	Definition
target	Optional. Administration Server only. A comma-separated list of server names, cluster names, or both from which active and available debug patches will be listed. The default value is `None`. If not specified, a list of debug patches only from the connected server will be displayed.

Example

The following example lists the active and available debug patches on the cluster `myCluster`.

The columns labeled `ms1` and `ms2` list the active and available patches deployed on Managed Servers `ms1` and `ms2`, respectively.

```
wls:/dyndebugDomain/serverConfig> listDebugPatches('myCluster')
ms1:
Active Patches:
    dyndebug01.jar:system
    dyndebug_app01.jar:app=myapp
    dyndebug_app02.jar:app=myapp
Available Patches:
    dyndebug00.jar
    dyndebug01.jar
    dyndebug02.jar
    dyndebug03.jar
    dyndebug04.jar
    dyndebug05.jar
    dyndebug_app01.jar
    dyndebug_app02.jar
    dyndebug_app03.jar

ms2:
Active Patches:
    dyndebug01.jar:system
    dyndebug_app01.jar:app=myapp
    dyndebug_app02.jar:app=myapp
Available Patches:
    dyndebug00.jar
    dyndebug01.jar
    dyndebug02.jar
    dyndebug03.jar
    dyndebug04.jar
    dyndebug05.jar
    dyndebug_app01.jar
    dyndebug_app02.jar
    dyndebug_app03.jar
```

listDebugPatchTasks

Command Category: Diagnostics Commands

Use with WLST: Online

Description

Lists the debug patch (activated or deactivated) tasks from the specified targets. While connected to the Administration Server, the optional `target` parameter can be specified to list tasks from multiple Managed Servers and clusters. If the `target` parameter is not specified, tasks from only the connected server will be listed.

Syntax

```
listDebugPatchTasks([target])
```

Argument	Definition
target	Optional. Administration Server only. A comma-separated list of server names, cluster names, or both from which debug patch tasks will be listed. The default value is None. If not specified, a list of debug patch tasks only from the connected server will be listed.

Example

The following example lists the debug patch tasks from the cluster `myCluster`.

The columns labeled `ms1` and `ms2` list the activated and deactivated patch tasks from Managed Servers `ms1` and `ms2`, respectively.

```
wls:/dyndebugDomain/serverConfig> tasks=listDebugPatchTasks(target='myCluster')
ms1:
  [1] DEACTIVATE_4 DEACTIVATE *  FINISHED
  [2] ACTIVATE_5 ACTIVATE dyndebug01.jar  FAILED
  [3] ACTIVATE_6 ACTIVATE dyndebug_app01.jar  FINISHED
  [4] DEACTIVATE_7 DEACTIVATE dyndebug01.jar  FINISHED


ms2:
  [1] DEACTIVATE_4 DEACTIVATE *  FINISHED
  [2] ACTIVATE_5 ACTIVATE dyndebug01.jar  FAILED
  [3] ACTIVATE_6 ACTIVATE dyndebug_app01.jar  FINISHED
  [4] DEACTIVATE_7 DEACTIVATE dyndebug01.jar  FINISHED
```

listSystemResourceControls

Command Category: Diagnostics Commands

Use with WLST: Online

Description

Lists all the System Resource Control MBeans that are available on the current server or specified targets.

Syntax

```
listSystemResourceControls([options]
```

Argument	Definition
options	**Server**—Administration Server only. The target server name from which system resource controls will be listed. The default value is `None`. This option has been replaced by the `Target` option; all WLST scripts should be updated to use `Target` instead. **Target**—A comma-separated list of server names, cluster names, or both from which system resource controls will be listed. The default value is `None`.

Example

The following example lists the diagnostic system resources deployed on `myServer` server.

- The column labeled `External` identifies whether the diagnostic system resource is defined by an external resource descriptor.
- The column labeled `Enabled` identifies whether the diagnostic resource is activated on the server on which it is configured.

```
wls:/mydomain/serverConfig> listSystemResourceControls('myServer')
External  Enabled   Name
--------  -------   ------------------------------
false     false     Module-0
false     false     allprofiles
true      true      MyExternal
false     true      Low
false     false     FPP-module
```

mergeDiagnosticData

Command Category: Diagnostics Commands

Use with WLST: Online

Description

Merges the set of data files previously generated by the `dumpDiagnosticData()` command and creates a single merged file in the directory specified by the *`inputDir`* argument. The data from all files in the specified directory is consolidated and written in CSV format.

Syntax

```
mergeDiagnosticData([inputDir], [options]
```

Argument	Definition
`inputDir`	The name of the source directory that contains the diagnostic data files.
`options`	**toFile**—The name of the target merged data file. The default is `merged.csv`.

Example

This example merges all the data files in the /home/mydir/data directory into a single data file called `mymergedata.csv`.

```
wls:/mydomain/serverRuntime>mergeDiagnosticData('/home/mydir/data',
toFile='mymergedata.csv')
```

purgeCapturedImages

Command Category: Diagnostics Commands

Use with WLST: Online

Description

Purges the diagnostic image files on the server as per the age criteria specified from the server's configured image destination directory.

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

Syntax

```
purgeCapturedImages([options])
```

Argument	Definition
options	**server**—Administration Server only. Name of the server from which the list of available images will be purged.If not specified, it defaults to `None`. **age**— Diagnostic images older than the specified age will be purged. Age is specified in the format `Days:Hours:Minutes`, where hours and minutes are optional. If the age is not specified, then all the existing image files will be purged.

Example

The following example purges images which are older than 15 days, 12 hours, 30 minutes from the Managed Server, `MS1`.

```
wls:/mydomain/serverConfig> purgeCapturedImages("15:12:30", server="MS1")
```

purgeDebugPatchTasks

Command Category: Diagnostics Commands

Use with WLST: Online

Description

Purges the debug patch (activated or deactivated) tasks on the specified targets. While connected to the Administration Server, the optional `target` parameter can be specified to purge tasks from multiple Managed Servers and clusters. If the `target` parameter is not specified, tasks from only the connected server are purged. Only completed, failed, or cancelled tasks are purged.

Syntax

```
purgeDebugPatchTasks([target])
```

Argument	Definition
target	Optional. Administration Server only. A comma-separated list of server names, cluster names, or both from which debug patch tasks are purged. The default value is `None`. If not specified, a list of debug patch tasks only from the connected server is purged.

Example

The following example purges debug patch tasks from the cluster `myCluster`.

```
wls:/dyndebugDomain/serverConfig> tasks=purgeDebugPatchTasks(target='myCluster')
```

showDebugPatchInfo

Command Category: Diagnostics Commands

Use with WLST: Online

Description

Displays details about a debug patch on the specified targets. While connected to the Administration Server, the optional `target` parameter can be specified to display debug patch information from multiple Managed Servers and clusters. The displayed information includes the content of the debug patch along with the content of the README file, if present in the debug patch.

Syntax

```
showDebugPatchInfo(patch, [target])
```

Argument	Definition
`patch`	Name of the debug patch for which information is displayed.
`target`	Optional. Administration Server only. A comma-separated list of server names, cluster names, or both. If not specified, then the debug patch information only from the connected server will be displayed. The default value is `None`.

Example

The following example displays information about the `dyndebug01.jar` debug patch that is deployed on the cluster `myCluster`.

The columns labeled `ms1` and `ms2` display information about the `dyndebug01.jar` debug patch deployed on Managed Servers `ms1` and `ms2`, respectively.

```
wls:/dyndebugDomain/serverConfig> showDebugPatchInfo('dyndebug01.jar',
target='myCluster'
ms1:
dyndebug01.jar:
    dyndebug.Class01
    dyndebug.Class02


Additional Information:
Sat Sep 13 14:24:17 EDT 2014

This is additional information for dyndebug01.



ms2:
dyndebug01.jar:
    dyndebug.Class01
    dyndebug.Class02


Additional Information:
Sat Sep 13 14:24:17 EDT 2014

This is additional information for dyndebug01.)
```

saveDiagnosticImageCaptureFile

Command Category: Diagnostics Commands

Use with WLST: Online

Description

Downloads the specified diagnostic image capture from the server to which WLST is currently connected.

See Configuring and Capturing Diagnostic Images in *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*.

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

Syntax

```
saveDiagnosticImageCaptureFile(imageName, [Server], [outputFile])
```

Argument	Definition
`imageName`	The name of the diagnostic image capture to download.
`options`	**Server**—Administration Server only. Name of the server from which the image is retrieved. Note that while connected to the Administration Server and retrieving an image from a Managed Server, the appropriate Server argument must be specified. If not specified, it defaults to `None`. **outputFile**—Local path and file name in which the retrieved diagnostic image capture is to be stored. If not specified, this argument defaults to the value of `imageName` and the current working directory. If not specified, it defaults to `None`.

Example

The following example retrieves the list of the diagnostic image captures that are stored in the image destination directory on the server. It then shows two uses of the `saveDiagnosticImageCaptureFile` command. In the first use, the first diagnostic image capture in the list is downloaded to the local machine using the default output file name. In the second use, the first diagnostic image capture in the list is downloaded to the local machine in the file `mylocalimg.zip`.

```
wls:/mydomain/serverRuntime>images=getAvailableCapturedImages()
Connecting to http://localhost:7001 with userid weblogic ...
wls:/mydomain/serverConfig> saveDiagnosticImageCaptureFile(images[0])
Retrieving diagnostic_image_myserver_2009_06_25_12_12_50.zip to local
path diagnostic_image_myserver_2009_06_25_12_12_50.zip
Connecting to http://localhost:7001 with userid weblogic ...
wls:/mydomain/serverConfig> saveDiagnosticImageCaptureFile(images[0],
'mylocalimg.zip')
Retrieving diagnostic_image_myserver_2009_06_25_12_12_50.zip to local
path mylocalimg.zip
Connecting to http://localhost:7001 with userid weblogic ...
```

saveDiagnosticImageCaptureEntryFile

Command Category: Diagnostics Commands

Use with WLST: Online

Description

Downloads a specific entry from the diagnostic image capture that is located on the server to which WLST is currently connected.

See Configuring and Capturing Diagnostic Images in *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*.

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

Syntax

```
saveDiagnosticImageCaptureEntryFile(imageName, imageEntryName, [Server], [outputFile])
```

Argument	Definition
`imageName`	Name of the diagnostic image capture containing the desired entry.

`imageEntryName`	Name of the specific entry to be retrieved from the diagnostic image capture. This can be one of the following:
	
	`image.summary`
	`JTA.xml`
	`FlightRecording.jfr`
	`WatchSource.xml`
	`configuration.zip`
	`WORK_MANAGER.txt`
	`JNDI_IMAGE_SOURCE.xml`
	`APPLICATION.xml`
	`InstrumentationImageSource.xml`
	`SAF.xml`
	`Logging.txt`
	`PERSISTENT_STORE.xml`
	`JDBC.txt`
	`PathService.xml`
	`JMS.xml`
	`Deployment.xml`
	`JVM.xml`
	`CONNECTOR.xml`

`Server`	Administration Server only. Server from which image entry is retrieved. Note that while connected to the Administration Server and retrieving an image entry from a Managed Server, the appropriate `Server` argument must be specified. If not specified, it defaults to `None`.
`outputFile`	Optional. Local path and file name in which the entry retrieved from the diagnostic image capture is to be stored. If not specified, this argument defaults to the value of `imageEntryName` and the current working directory.If not specified, it defaults to `None`.

Example

The following example gets the list of diagnostic image captures, then uses the `saveDiagnosticImageCaptureEntryFile` command twice. In the first use, this command retrieves the image summary to the local machine using the default output file name. In the second use, it retrieves the image summary to the local machine in the file `myimage.summary`.

```
wls:/mydomain/serverRuntime>images=getAvailableCapturedImages()
Connecting to http://localhost:7001 with userid weblogic ...
wls:/mydomain/serverConfig> saveDiagnosticImageCaptureEntryFile(images[0],
'image.summary')
Retrieving entry image.summary from
diagnostic_image_myserver_2009_06_25_12_12_50.zip to local path image.summary
Connecting to http://localhost:7001 with userid weblogic ...
wls:/mydomain/serverConfig> saveDiagnosticImageCaptureEntryFile(images[0],
'image.summary', 'myimage.summary')
Retrieving entry image.summary from
diagnostic_image_myserver_2009_06_25_12_12_50.zip to local path myimage.summary
Connecting to http://localhost:7001 with userid weblogic ...
```

Editing Commands

Use the WLST editing commands to interrogate and edit configuration beans. Table 2-7 lists and summarizes these commands.

Note:

To edit configuration beans, you must be connected to an Administration Server, and you must navigate to the edit tree and start an edit session, as described in edit and startEdit, respectively.

If you connect to a Managed Server, WLST functionality is limited to browsing the configuration bean hierarchy. While you cannot use WLST to change the values of MBeans on Managed Servers, it is possible to use the Management APIs to do so. Oracle recommends that you change only the values of configuration MBeans on the Administration Server. Changing the values of MBeans on Managed Servers can lead to an inconsistent domain configuration.

See Using WLST Online to Update an Existing Domain in *Understanding the WebLogic Scripting Tool*.

Table 2-7 Editing Commands for WLST Configuration

This command...	Enables you to...	Use with WLST...
activate	Activate changes saved during the current editing session but not yet deployed.	Online or Offline
assign	Assign resources to one or more destinations.	Offline
cancelEdit	Cancel an edit session, release the edit lock, and discard all unsaved changes. This operation can be called by any user with administrator privileges, even if the user did not start the edit session.	Online
create	Create a configuration bean of the specified type for the current bean.	Online or Offline
createEditSession	Create a new edit session.	Online

Table 2-7 (Cont.) Editing Commands for WLST Configuration

This command...	Enables you to...	Use with WLST...
delete	Delete an instance of a configuration for the current configuration bean.	Online or Offline
deleteFEHost	Delete the Frontend host for a domain.	Offline
destroyEditSession	Remove the specified edit session.	Online
enableOverwriteComponentChanges	Overwrite changes to all system components during activation.	Online
encrypt	Encrypt the specified string.	Online
get	Return the value of the specified attribute.	Online or Offline
getActivationTask	Return the latest `ActivationTask` MBean on which a user can get status.	Online
invoke	Invokes a management operation on the current configuration bean.	Online
isRestartRequired	Determine whether a server restart is required.	Online
loadDB	Load SQL files into a database.	Offline
loadProperties	Load property values from a file.	Online or Offline
pullComponentChanges	Pull the configuration changes of a system component from the remote node.	Online
resolve	Detect any external modification or conflict, and resolve them.	Online
resync	Resynchronize the configuration files of the specified system component.	Online
resyncAll	Resynchronize the configuration files of all system components.	Online
save	Save the edits that have been made but have not yet been saved.	Online
set	Set the specified attribute value for the current configuration bean.	Online or Offline
setOption	Set options related to a WebLogic domain creation or update.	Offline
showChanges	Show the changes made to the configuration by the current user during the current edit session.	Online
showComponentChanges	Show the changes to the configuration of the specified system component on the remote node.	Online
showEditSession	Show information about the specified edit sessions.	Online
startEdit	Starts a configuration edit session on behalf of the currently connected user.	Online
stopEdit	Stop the current edit session, release the edit lock, and discard unsaved changes.	Online
unassign	Unassign applications or resources from one or more destinations.	Offline

Table 2-7 (Cont.) Editing Commands for WLST Configuration

This command...	Enables you to...	Use with WLST...
undo	Revert all unsaved or unactivated edits.	Online
validate	Validate the changes that have been made but have not yet been saved.	Online

activate

Command Category: Editing Commands

Use with WLST: Online

Description

Activates changes saved during the current editing session but not yet deployed. This command prints a message if a server restart is required for the changes that are being activated.

Multiple edit sessions with different names may coexist in the system. While making changes on a named edit session, a user of another edit session can concurrently make configuration modifications and activate those changes. To prevent inconsistent modifications, the system checks for any external configuration changes before every activation. Any non-conflicting concurrent modification is automatically merged. In case of an unresolved conflict, the system terminates the activation and returns an error. To list all existing edit sessions, use the `showEditSession()` command. See showEditSession. To resolve potential modification conflicts, use the `resolve()` command. See resolve.

The activate command returns the latest `ActivationTask` MBean, which reflects the state of changes that a user is currently making or has made recently. You can then invoke methods to get information about the latest Configuration Manager activate task in progress or just completed. In the event of an error, the command returns a `WLSTException`.

Note the following locking mechanism behaviors when using `activate`:

- When running concurrent edit sessions under the same user account using an exclusive lock, activating changes using `activate()` in one session releases the lock for that session, allowing the next session to obtain the lock. Therefore, there is no need to use `cancelEdit()` after `activate()` in the first session to release the lock. Doing so will actually cancel the edit session for the next session in the queue, potentially causing an exception.
- There is only ever one lock in a domain and only a single user can have it at any given time. Unless it is an exclusive lock, that user can acquire a lock from any number of sessions. Each session will add its changes to the existing set of changes. When `activate()` is called in any session, all changes made up to that point by all sessions are activated. Similarly, when `cancelEdit()` is called in any session, all changes made up to that point by all sessions are cancelled, the lock will be released by that user, and an edit lock can then be acquired by another user.

Syntax

```
activate([timeout], [block])
```

Argument	Definition
`timeout`	Optional. Time (in milliseconds) that WLST waits for the activation of configuration changes to complete before canceling the operation. A value of -1 indicates that the operation will not time out. This argument defaults to 300,000 ms (or 5 minutes).
`block`	Optional. Boolean value specifying whether WLST should block user interaction until the command completes. This argument defaults to `false`, indicating that user interaction is not blocked. In this case, WLST returns control to the user after issuing the command and assigns the task MBean associated with the current task to a variable that you can use to check its status. If you are importing WLST as a Jython module, as described in Importing WLST as a Jython Module in *Understanding the WebLogic Scripting Tool*, *`block`* is always set to `true`.

Example

The following example activates the changes made during the current edit session that have been saved to disk, but that have not yet been activated. The WLST waits 200,000 ms for the activation of configuration changes to complete.

```
wls:/mydomain/edit !> activate(200000, block='true')
Activating all your changes, this may take a while ...
The edit lock associated with this edit session is released once the activation is
completed.
Action completed.
wls:/mydomain/edit>
```

assign

Command Category: Editing Commands

Use with WLST: Offline

Description

Assigns resources to one or more destinations.

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

Syntax

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

Argument	Definition
sourceType	Type of configuration bean to be assigned. This value can be set to one of the following values: • `AppDeployment` • `Library` • `securityType (such as User)` • `Server` • `service (such as JDBCSystemResource)` • *`service`*`.SubDeployment`, where *`service`* specifies the service type of the `SubDeployment` (such as `JMSSystemResource.SubDeployment`); you can also specify nested subdeployments (such as `AppDeployment.SubDeployment.SubDeployment`) Guidelines for setting this value are provided below.
sourceName	Name of the resource to be assigned. Multiple names can be specified, separated by commas, or you can use the wildcard (*) character to specify all resources of the specified type. Specify subdeployments using the following format: *`service.subDeployment`*, where *`service`* specifies the parent service and *`subDeployment`* specifies the name of the subdeployment. For example, `myJMSResource.myQueueSubDeployment`. You can also specify nested subdeployments, such as `MedRecEAR.MedRecAppScopedJMS.MedRecJMSServer`. **Note:** A given subdeployment name cannot contain a dot (.), as the `assign` command will interpret it as a nested subdeployment.
destinationType	Type of destination. Guidelines for setting this value are provided below.
destinationName	Name of the destination. Multiple names can be specified, separated by commas. To specify a migratable target, such as when assigning JMS services to a migratable Managed Server, include `migratable` after the destination name. For example: `managed1 (migratable)`

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

- When assigning **application deployments**, set the values as follows:
  - *`sourceType`*: `AppDeployment`
  - *`destinationType`*: `Target`
- When assigning **libraries**, set the values as follows:
  - *`sourceType`*: `Library`
  - *`destinationType`*: `Target`
- When assigning **services**, set the values as follows:
  - *`sourceType`*: Name of the specific server, such as `JDBCSystemResource`
  - *`destinationType`*: `Target`
- When assigning **servers** to **clusters**, set the values as follows:

  - *sourceType*: `Server`
  - *destinationType*: `Cluster`
- When assigning **subdeployments**, set the values as follows:
  - *sourceType*: *`service`*`.SubDeployment`, where *`service`* specifies the parent of the `SubDeployment`, such as `JMSSystemResource.SubDeployment`; you can also specify nested subdeployments (such as `AppDeployment.SubDeployment.SubDeployment`)
  - *destinationType*: `Target`
- When assigning **security types**, set the values as follows:
  - *sourceType*: Name of the security type, such as `User`
  - *destinationType*: Name of the destination security type, such as `Group`

Example

The following examples:

- Assign the servers `myServer` and `myServer2` to the cluster `myCluster`.

  ```
  wls:/offline/mydomain> assign("Server", "myServer,myServer2", "Cluster",
  "myCluster")
  ```

- Assign all servers to the cluster `myCluster`.

  ```
  wls:/offline/mydomain> assign("Server", "*", "Cluster", "myCluster")
  ```

- Assign the application deployment `myAppDeployment` to the target server `newServer`.

  ```
  wls:/offline/mydomain> assign("AppDeployment", "myAppDeployment", "Target",
  "newServer")
  ```

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

  ```
  wls:/offline/mydomain> assign("User", "newUser", "Group", "Monitors")
  ```

- Assign the SubDeployment `myQueueSubDeployment`, which is a child of the JMS resource `myJMSResource`, to the target server `newServer`.

  ```
  wls:/offline/mydomain> assign('JMSSystemResource.SubDeployment',
  'myJMSResource.myQueueSubDeployment', 'Target', 'newServer')
  ```

- Assign the nested SubDeployment `MedRecAppScopedJMS.MedRecJMSServer`, which is a child of the AppDeployment `AppDeployment`, to the target server `AdminServer`.

  ```
  wls:/offline/mydomain>assign('AppDeployment.SubDeployment.SubDeployment
  ','MedRecEAR.MedRecAppScopedJMS.MedRecJMSServer','Target','AdminServer')
  ```

- Assign the JMS file store `myFileStore` to a migratable target Managed Server called `managed1`.

  ```
  wls:/offline/mydomain>assign('FileStore', 'myFileStore', 'Target',
  'managed1 (migratable)'
  ```

cancelEdit

Command Category: Editing Commands

Use with WLST: Online

Description

Cancels an edit session, releases the edit lock, and discards all unsaved changes.

The user issuing this command does not have to be the current editor; this allows an administrator to cancel an edit session, if necessary, to enable other users to start an edit session.

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

For additional information about `cancelEdit`, see the activate command.

Syntax

```
cancelEdit([defaultAnswer])
```

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

Example

The following example cancels the current editing session. WLST prompts for verification before canceling.

```
wls:/mydomain/edit !> cancelEdit()
Sure you would like to cancel the edit session? (y/n)y
Edit session is cancelled successfully
wls:/mydomain/edit>
```

create

Command Category: Editing Commands

Use with WLST: Online or Offline

Description

Creates a configuration bean of the specified type for the current bean.

The `create` command returns a stub for the newly created configuration bean. In the event of an error, the command returns a `WLSTException`.

Note:

Child types must be created under an instance of their parent type. You can only create configuration beans that are children of the current Configuration Management Object (`cmo`) type. See Changing the Current Management Object in *Understanding the WebLogic Scripting Tool*.

Please note the following when using the `create` command with **WLST online**:

- You must be connected to an Administration Server. You cannot use the `create` command for runtime MBeans or when WLST is connected to a Managed Server instance.
- You must navigate to the edit configuration MBean hierarchy using the `edit` command before issuing this command. See edit.
- You can use the create command to create a WebLogic Server configuration MBean that is a child of the current MBean type.

Please note the following when using the `create` command with **WLST offline**:

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

To know more about:

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

Syntax

```
create(name, childMBeanType, [baseProviderType])
```

Argument	Definition
`name`	Name of the configuration bean that you are creating.
`childMBeanType`	Type of configuration bean that you are creating. You can create instances of any type defined in the `config.xml` file except custom security types. See *MBean Reference for Oracle WebLogic Server*.
`baseProviderType`	When creating a security provider, specifies the base security provider type, for example, `AuthenticationProvider`. This argument defaults to None.

Example

The following example creates a child configuration bean of type `Server` named newServer for the current configuration bean, storing the stub as `server1`:

```
wls:/mydomain/edit !> server1=create('newServer','Server')
Server with name 'newServer' has been created successfully.
wls:/mydomain/edit !> server1.getName()
'newServer'
wls:/mydomain/edit !>
```

The following example creates an authentication provider security provider called `myProvider`:

```
wls:/mydomain/edit !> cd('SecurityConfiguration/mydomain/Realms/myrealm')
wls:/mydomain/edit !>
create('myProvider','weblogic.security.providers.authentication.SQLAuthenticator'
,'AuthenticationProvider')
wls:/mydomain/edit ! cd('AuthenticationProviders/myProvider')
wls:/mydomain/edit ! set('ControlFlag', 'REQUIRED')
```

The following example creates a machine named `highsec_nm` and sets attributes for the associated Node Manager.

```
wls:/mydomain/edit !> create('highsec_nm', 'Machine')
wls:/mydomain/edit !> cd('Machine/highsec_nm/NodeManager/highsec_nm')
wls:/mydomain/edit !> set('DebugEnabled', 'true')
wls:/mydomain/edit !> set('ListenAddress', 'innes')
wls:/mydomain/edit !> set('NMType', 'SSL')
wls:/mydomain/edit !> set('ShellCommand', '')
```

createEditSession

Command Category: Editing Commands

Use with WLST: Online

Description

Creates a new named edit session with the specified name and description. To navigate to the context of the created edit session, use the `edit(editSessionName)` command.

The creation of a new edit session may fail in case a session with the specified name already exists, or if the specified edit session name contains an invalid character.

Syntax

```
createEditSession(name, [description])
```

Argument	Definition
name	Name or identifier of an edit session. It can contain alphanumeric characters, an underscore (_), and a dash (-).
description	Optional. Description of the edit session for easy identification by a user.

Example

The following example creates an edit session named `mySampleSession` and provides a useful description of the session to be created:

```
wls:/wls/serverConfig> createEditSession('mySampleSession', 'This session is
created to test this command')
```

delete

Command Category: Editing Commands

Use with WLST: Online or Offline

Description

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

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

Note:

You can only delete configuration beans that are children of current Configuration Management Object (`cmo`) type. See Changing the Current Management Object in *Understanding the WebLogic Scripting Tool*.

Syntax

```
delete(name, childMBeanType)
```

Argument	Definition
`name`	Name of the child configuration bean to delete.
`childMBeanType`	Type of the configuration bean to be deleted. You can delete instances of any type defined in the `config.xml` file. See *MBean Reference for Oracle WebLogic Server*.

Example

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

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

deleteFEHost

Command Category: Editing Commands

Use with WLST: Offline

Description

Deletes the plain, SSL and default URL values for the domain Frontend Host from the service table.

Syntax

```
deleteFEHost()
```

Example

The following example deletes the domain Frontend Host URL values for the domain mydomain.

```
wls:/offline> readDomain("/domains/mydomain")
wls:/offline> deleteFEHost()
```

destroyEditSession

Command Category: Editing Commands

Use with WLST: Online

Description

Removes or destroys the specified edit session.

It is not possible to remove the default (nameless) edit session. An edit session that contains changes that are pending activation or are currently being activated cannot be destroyed by default. You must set the force flag to `true` if you want to destroy a session with pending activation changes, or if the owner of the specified edit session is different from the current user.

Syntax

```
destroyEditSession(name, [force])
```

Argument	Definition
name	Name or identifier of the edit session to be removed.
force	Optional. If set to `true`, then the session with pending activation changes is removed. If set to `false`, then the session with pending activation changes is preserved, and the command fails. This argument defaults to `false`.

Example

The following example removes or destroys the edit session named `four`. The force flag is set to `true`; therefore, if the specified edit session contains pending activation changes, then the session is removed or destroyed.

```
wls:/wls/serverConfig> destroyEditSession('four', 'true')
```

enableOverwriteComponentChanges

Command Category: Editing Commands

Use with WLST: Online

Description

Overwrites changes to all system components during activation. This command requires an active edit session.

Syntax

```
enableOverwriteComponentChanges()
```

Example

The following example overwrites the changes to all system components during activation:

```
startEdit()

enableOverwriteComponentChanges()
```

encrypt

Command Category: Editing Commands

Use with WLST: Online

Description

Encrypts the specified string. You can then use the encrypted string in your configuration file or as an argument to a command.

You must invoke this command once for each WebLogic domain in which you want to use the encrypted string. The string can be used only in the WebLogic domain for which it was originally encrypted.

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

Syntax

```
encrypt(obj, [domainDir])
```

Argument	Definition
obj	String that you want to encrypt.
domainDir	Optional. Absolute path name of a WebLogic domain directory. The encrypted string can be used only by the WebLogic domain that is contained within the specified directory. If you do not specify this argument, the command encrypts the string for use in the WebLogic domain to which WLST is currently connected.

Example

The following example encrypts the specified string using the `security/SerializedSystemIni.dat` file in the specified WebLogic domain directory.

```
wls:/mydomain/serverConfig> es=encrypt('password','c:/Oracle/Middleware/domains/
mydomain')
```

get

Command Category: Editing Commands

Use with WLST: Online or Offline

Description

Returns the value of the specified attribute. For more information about the MBean attributes that can be viewed, see *Oracle WebLogic Server MBean Reference*. In the event of an error, the command returns a `WLSTException`.

Note:

You can list all attributes and their current values by entering `ls('a')`. See ls.

Alternatively, you can use the `cmo` variable to perform any get method on the current configuration bean. For example:

```
cmo.getListenPort()
```

See Changing the Current Management Object in *Understanding the WebLogic Scripting Tool*.

Syntax

```
get(attrName)
```

Argument	Definition
attrName	Name of the attribute to be displayed. You can specify the full pathname of the attribute. If no pathname is specified, the attribute is displayed for the current configuration object.

Example

The following example returns the value of the `AdministrationPort` for the current configuration bean.

```
wls:/mydomain/serverConfig> get('AdministrationPort')
9002
```

Alternatively, you can use the `cmo` variable:

```
cmo.getAdministrationPort()
```

getActivationTask

Command Category: Editing Commands

Use with WLST: Online

Description

Return the latest `ActivationTask` MBean on which a user can get status. The `ActivationTask` MBean reflects the state of changes that a user has made recently in WLST. You can then invoke methods to get information about the latest Configuration Manager activate task in progress or just completed. In the event of an error, the command returns a `WLSTException`.

Note:

If you have activated changes outside of WLST, use the `ConfigurationManagerMBean getActivationTasks()` method to get access to Activation Tasks created in other tools.

Syntax

```
getActivationTask()
```

Example

The following example returns the latest `ActivationTask` MBean on which a user can get status and stores it within the task variable.

```
wls:/mydomain/edit> task=getActivationTask()
wls:/mydomain/edit> if task!=None:
...    task.getState()
...
4
```

invoke

Command Category: Editing Commands

Use with WLST: Online

Description

Invokes a management operation on the current configuration bean. Typically, you use this command to invoke operations other than the `get` and `set` operations that most WebLogic Server configuration beans provide. The class objects are loaded through the same class loader that is used for loading the configuration bean on which the action is invoked.

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

If successful, the `invoke` command returns the object that is returned by the operation invoked. In the event of an error, the command returns a `WLSTException`.

Syntax

```
invoke(methodName, parameters, signatures)
```

Argument	Definition
methodName	Name of the method to be invoked.
parameters	An array of parameters to be passed to the method call.
signatures	An array containing the signature of the action.

Example

The following example invokes the `lookupServer` method on the current configuration bean.

```
wls:/mydomain/config> objs =
jarray.array([java.lang.String("oamserver")],java.lang.Object)
wls:/mydomain/edit> strs = jarray.array(["java.lang.String"],java.lang.String)
wls:/mydomain/edit> invoke('lookupServer',objs,strs)
true
wls:/mydomain/edit>
```

isRestartRequired

Command Category: Editing Commands

Use with WLST: Online

Description

Determines whether a server restart is required.

If you invoke this command while an edit session is in progress, the response is based on the edits that are currently in progress. If you specify the name of an attribute, WLST indicates whether a server restart is required for that attribute only.

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

Syntax

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

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

Example

The following example specifies whether a server restart is required for all changes made during the current WLST session.

```
wls:/mydomain/edit !> isRestartRequired()
Server re-start is REQUIRED for the set of changes in progress.

The following attribute(s) have been changed on MBeans that require server re-
start.
MBean Changed : mydomain:Name=mydomain,Type=Domain
Attributes changed : AutoConfigurationSaveEnabled
```

The following example specifies whether a server restart is required if you edit the `ConsoleEnabled` attribute.

```
wls:/mydomain/edit !> isRestartRequired("ConsoleEnabled")
Server re-start is REQUIRED if you change the attribute ConsoleEnabled wls:/
mydomain/edit !>
```

loadDB

Command Category: Editing Commands

Use with WLST: Offline

Description

Loads SQL files into a database.

The `loadDB` command loads the SQL files from a template file. This command can only be issued after a domain template or extension template has been loaded into memory (see readDomain and readTemplate).

Before executing this command, ensure that the following conditions are true:

- The appropriate database is running.
- SQL files exist for the specified database and version.

  To verify that the appropriate SQL files exist, open the domain template and locate the relevant SQL file list, `jdbc.index`, in the `_jdbc_` directory. For example, for Oracle 9*i*, the SQL file list is located at `_jdbc_\Oracle\9i\jdbc.index`.

The command fails if the above conditions are not met.

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

Syntax

```
loadDB(dbVersion, datasourceName, dbCategory)
```

Argument	Definition
`dbVersion`	Version of the database for which the SQL files are intended to be used.
`datasourceName`	Name of the JDBC data source to be used to load SQL files.
`dbCategory`	Optional. Database category associated with the specified data source. See Files Typically Included in a Template in the *Oracle WebLogic Server Domain Template Reference*.

Example

The following example loads SQL files related to `Drop/Create P13N Database Objects` intended for version `5.1` of the database, using the `p13nDataSource` JDBC data source.

```
wls:/offline/mydomain> loadDB('5.1', 'p13nDataSource', 'Drop/Create P13N Database
Objects')
```

loadProperties

Command Category: Editing Commands

Use with WLST: Online and Offline

Description

Loads property values from a file and makes them available in the WLST session.

This command cannot be used when you are importing WLST as a Jython module, as described in Importing WLST as a Jython Module in *Understanding the WebLogic Scripting Tool*.

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

Syntax

```
loadProperties(fileName)
```

Argument	Definition
`fileName`	Properties file pathname.

Example

This example gets and sets the properties file values.

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

pullComponentChanges

Command Category: Editing Commands

Use with WLST: Online

Description

Pulls the configuration changes of a system component from the remote node to the current edit session. This command requires an active edit session.

Syntax

```
pullComponentChanges(compName)
```

Argument	Definition
compName	Name of the system component for which to pull the configuration changes.

Example

The following example starts an edit session, and pulls the configuration changes of the system component, `otd_test_varunam.in.oracle.com`, from the remote node to the current edit session. It then activates the changes.

```
startEdit()

pullComponentChanges('otd_test_varunam.in.oracle.com')
pull component otd_test_varunam.in.oracle.com changes on machine
varunam.in.oracle.com:
add OTD/test/config/foo.conf
edit OTD/test/config/server.xml
edit OTD/test/config/test-obj.conf
remove OTD/test/config/obj.conf

activate()
```

resolve

Command Category: Editing Commands

Use with WLST: Online

Description

Detects conflicts between changes made as part of the current edit session and changes that are activated externally. In case one or more conflicts are found, the `stopOnConflicts` parameter determines whether the conflicts are either automatically resolved using a set of predefined rules, or the list of conflicts is displayed and the command execution is terminated. This command works with the current edit session. In order to use this command, the user must obtain the edit session lock.

Once all conflicts are resolved, the command updates the current edit session with any non-conflicting changes made externally since the session creation or since the last successful call to `resolve()`. The `resolve()` command only updates the content of the current edit session. Any pending changes in the edit session are not activated.

The `resolve()` command always displays the list of found conflicts, description of resolve operation, and the list of all external changes applied to this edit session as part of the `resolve()` invocation.

Syntax

```
resolve([stopOnConflicts])
```

Argument	Definition
stopOnConflicts	Optional. A boolean flag that determines the resolution in case one or more conflicts are detected. If set to `false`, the command tries to resolve the conflicts automatically. If set to `true`, the command lists all conflicts found and terminates the command execution. This argument defaults to `false`.

Example

The following example calls the `resolve()` command, which in turn lists a conflict that is found. This command provides a description of the conflict and also of the `resolve` operation:

```
wls:/wls/edit(one)/Servers/s1 !> resolve()
      1 conflict:
      [1]
      [wls]/Servers[s1] - A conflict in a property ListenPort has been detected.
      Original value: 7001
      Edit value: 8111
      Runtime value: 8222
      Description of resolve operation:
      The changes made to ListenPort by the current edit session will override the
ones present in the current runtime configuration.

      Patch:
      No difference.
```

resync

Command Category: Editing Commands

Use with WLST: Online

Description

Resynchronizes the configuration files of the specified system component from the Administration Server to the corresponding remote node. This command must be invoked outside the edit session.

Note:

This command automatically starts an edit session to prevent configuration changes to occur during the resynchronization.

Syntax

```
resync(compName)
```

Argument	Definition
compName	Name of the system component for which to resynchronize the configuration files.

Example

The following example shows the configuration file changes for the system component, `otd_test_varunam.in.oracle.com`. It then resynchronizes the configuration files for this system component.

```
showComponentChanges('otd_test_varunam.in.oracle.com')
add OTD/test/config/bar.conf 1970.01.01-05:30:00 2014.11.07-17:35:15
edit OTD/test/config/proxyvs.obj.conf 2014.11.07-17:36:49 1970.01.01-05:29:59
edit OTD/test/config/server.xml 2014.11.07-17:36:49 2014.11.07-17:37:22
remove OTD/test/config/test-obj.conf 2014.11.07-17:36:49 1970.01.01-05:30:00

resync('otd_test_varunam.in.oracle.com')

showComponentChanges('otd_test_varunam.in.oracle.com')
component otd_test_varunam.in.oracle.com changes on machine
varunam.in.oracle.com: no change found.
```

resyncAll

Command Category: Editing Commands

Use with WLST: Online

Description

Resynchronizes the configuration files of all system components.

Syntax

```
resyncAll()
```

Example

The following example shows the configuration changes for all system components. It then resynchronizes the configuration files for all system components.

```
>showComponentChanges()
component otd_test_varunam.in.oracle.com changes on machine
varunam.in.oracle.com:
add OTD/test/config/baz.conf 1970.01.01-05:30:00 2014.11.07-17:42:57
component otd_origin-server-1_varunam.in.oracle.com changes on machine
varunam.in.oracle.com:
add OTD/origin-server-1/config/bar.conf 1970.01.01-05:30:00 2014.11.07-17:43:34

resyncAll()

showComponentChanges()
component otd_test_varunam.in.oracle.com changes on machine
varunam.in.oracle.com: no change found.
component otd_origin-server-1_varunam.in.oracle.com changes on machine
varunam.in.oracle.com: no change found.
```

save

Command Category: Editing Commands

Use with WLST: Online

Description

Saves the edits that have been made but have not yet been saved. This command is only valid when an edit session is in progress. For information about starting an edit session, see startEdit.

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

Syntax

```
save()
```

Example

The following example saves the edits that have not yet been saved to disk.

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

set

Command Category: Editing Commands

Use with WLST: Online or Offline

Description

Sets the value of a specified attribute in the current management object. When using WLST offline, this command writes the attribute value to the domain configuration files. When using WLST online, this command sets the value of an MBean attribute. Online changes are written to the domain configuration file when you activate your edits.

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

For information about setting encrypted attributes (all encrypted attributes have names that end with `Encrypted`), see Writing and Reading Encrypted Configuration Values in *Understanding the WebLogic Scripting Tool*.

Note the following when using **WLST online**:

- You must be in an edit session to use this command. See startEdit.
- You cannot use this command when WLST is connected to a Managed Server.
- As an alternative to this command, you can use the `cmo` variable with the following syntax:

  ```
  cmo.setattrName(value)
  ```

  For example, instead of using `set('ListenPort', 7011)`, you can use:

  ```
  cmo.setListenPort(7011)
  ```

See Changing the Current Management Object in *Understanding the WebLogic Scripting Tool*.

Syntax

```
set(attrName, value)
```

Argument	Definition
`attrName`	Name of the attribute to be set.
`value`	Value of the attribute to be set. **Note:** This value should *not* be enclosed in single or double quotes. See the examples.

Example

The following example sets the `ArchiveConfigurationCount` attribute of `DomainMBean` to `10`:

```
wls:/mydomain/serverConfig> set('ArchiveConfigurationCount', 10)
```

The following example sets the long value of the `T1TimerInterval` attribute of a custom Mbean to `123`:

```
wls:/mydomain/serverConfig> set('T1TimerInterval', Long(123))
```

The following example sets the boolean value of the `MyBooleanAttribute` attribute of a custom Mbean `to` `true`:

```
wls:/mydomain/serverConfig> set('MyBooleanAttribute', Boolean(true))
```

setOption

Command Category: Editing Commands

Use with WLST: Offline

Description

Sets options related to a WebLogic domain creation or update. In the event of an error, the command returns a `WLSTException`.

Syntax

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

Argument	Definition
`optionName`	Name of the option to set. See the following table for possible values.
`optionValue`	Value for the option. **Note:** Boolean values can be specified as a String (`true`, `false`) or integer (`0`, `1`).

Option Name	Used for	Description
`DomainName`	domain creation	Name of the WebLogic domain. By default, the name of the WebLogic domain is derived from the name of the domain directory. For example, for a WebLogic domain saved to `c:/Oracle/Middleware/user_projects/domains/myMedrec`, the domain name is `myMedrec`. By setting `DomainName`, the name of the created domain will be independent of the domain directory name.
`JavaHome`	domain creation	Home directory for the JVM to be used when starting the server. The default for this option depends on the platform on which you install WebLogic Server.
`NodeManagerType`	domain creation	The Node Manager type to be used to create a Node Manager for the domain. This option defaults to `PerDomainNodeManager`. • `PerDomainNodeManager`—if you specify this option, the Node Manager home is predefined within the domain as `<domain_name>/nodemanager` and you cannot edit the Node Manager home. The Node Manager for each domain can have a different configuration, as determined by the files in this directory. • `CustomLocationNodeManager`—specify this option if you want the per-domain Node Manager configuration files to be created in a specific location. The specified directory must be empty. • `ManualNodeManagerSetup`—specify this option if you want to manually set up the Node Manager configuration. If selected, the Node Manager configuration for the domain is ignored, and you must manually configure any required changes to the existing Node Manager configuration. **Note:** When upgrading a domain and changing from a per-host Node Manager configuration to a per-domain Node Manager configuration, if you are using custom scripts to start and stop the WebLogic Server environment, you must manually update the scripts to change the Node Manager home location to the new domain-based location. See Configuring Java Node Manager in *Administering Node Manager for Oracle WebLogic Server*. See Default Node Manager Configuration in *Administering Node Manager for Oracle WebLogic Server*
`NodeManagerHome`	domain creation	Node Manager home is the Node Manager directory to be created for the domain. This option is used only when `CustomLocationNodeManager` is specified for `NodeManagerType`.
`OldNodeManagerHome`	domain reconfiguration	A Node Manager home directory from which the existing configuration is taken for Node Manager upgrade during domain reconfiguration. This option applies only when the Node Manager upgrade type is `Migrate`.
`NodeManagerUpgradeType`	domain reconfiguration	Node Manager upgrade can be `New` or `Migrate`. `New` does not use an existing Node Manager configuration and creates the specified type of Node Manager. `Migrate` migrates the Node Manager configuration from the Node Manager home specified by the `OldNodeManagerHome` option. This option is used for upgrading Node Manager during domain reconfiguration when the Node Manager type is either `PerDomainNodeManager` or `CustomLocationNodeManager`.

Option Name	Used for	Description
`NodeManagerUpgradeOverwriteDefault`	domain reconfiguration	A boolean flag that is used to overwrite the Oracle-recommended default values for mandatory Node Manager configuration. This option is used only when Node Manager upgrade type is `Migrate`. The default value is `false`.
`OverwriteDomain`	domain creation	Boolean value specifying whether to allow an existing WebLogic domain to be overwritten. This option defaults to `false`.
`ServerStartMode`	domain creation	Mode to use when starting the server for the newly created WebLogic domain. This value can be `dev` (development), `prod` (production), or `secure` (secured production). This option defaults to `dev`.
`AllowCasualUpdate`	domain upgrade	Boolean value specifying whether to allow a WebLogic domain to be updated without adding an extension template. This option defaults to `true`.
`ReplaceDuplicates`	domain upgrade	Boolean value specifying whether to keep original configuration elements in the WebLogic domain or replace the elements with corresponding ones from an extension template when there is a conflict. This option defaults to `true`.
`AppDir`	domain creation and domain upgrade	Application directory to be used when a separate directory is desired for applications, as specified by the template. This option defaults to *`WL_HOME`*`/user_projects/applications/`*`domainname`*, where *`WL_HOME`* specifies the WebLogic Server home directory and *`domainname`* specifies the name of the WebLogic domain.
`AutoAdjustSubDeploymentTarget`	domain creation and domain upgrade	Boolean value specifying whether WLST automatically adjusts targets for the subdeployments of AppDeployments. This option defaults to `true`. To deactivate this feature, set the option to `false` and explicitly set the targeting for AppDeployment subdeployments before writing or updating the WebLogic domain or domain template.
`AutoDeploy`	domain creation and domain upgrade	Boolean value specifying whether to activate auto deployment when a cluster or multiple Managed Servers are created. This option defaults to `true`. To deactivate this feature, set the option to `false` on the first line of your script.

Example

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

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

showChanges

Command Category: Editing Commands

Use with WLST: Online

Description

Shows the changes made to the configuration by the current user during the current edit session. In the event of an error, the command returns a `WLSTException`.

Syntax

```
showChanges([onlyInMemory])
```

Argument	Definition
onlyInMemory	Optional. Boolean value specifying whether to display only the changes that have not yet been saved. This argument defaults to `false`, indicating that all changes that have been made from the start of the session are displayed.

Example

The following example shows all of the changes made by the current user to the configuration since the start of the current edit session.

```
wls:/mydomain/edit !> showChanges()
Changes that are in memory and saved to disc but not yet activated are:

MBean Changed           : com.bea:Name=basicWLSDomain,Type=Domain
Operation Invoked       : add
Attribute Modified      : Machines
Attributes Old Value    : null
Attributes New Value    : Mach1
Server Restart Required : false

MBean Changed           : com.bea:Name=basicWLSDomain,Type=Domain
Operation Invoked       : add
Attribute Modified      : Servers
Attributes Old Value    : null
Attributes New Value    : myserver
Server Restart Required : false
```

showComponentChanges

Command Category: Editing Commands

Use with WLST: Online

Description

Shows changes to the configuration of the specified system component on the remote node in comparison to the Administration Server.

Syntax

```
showComponentChanges([compName])
```

Argument	Definition
compName	Optional. Name of the system component for which to compare the configuration changes. If not specified, then the comparison is performed for all system components in the domain.

Example

The following example shows configuration changes for all system components in the current domain.

```
showComponentChanges()
component otd_test_varunam.in.oracle.com changes on machine
varunam.in.oracle.com: no change found.
component otd_origin-server-1_varunam.in.oracle.com changes on machine
varunam.in.oracle.com: no change found.
component otd_origin-server-2_varunam.in.oracle.com changes on machine
varunam.in.oracle.com: no change found.
component otd_origin-server-3_varunam.in.oracle.com changes on machine
varunam.in.oracle.com: no change found.
```

The following example shows configuration changes for the system component, `otd_test_varunam.in.oracle.com`.

```
showComponentChanges('otd_test_varunam.in.oracle.com')
add OTD/test/config/foo.conf 1970.01.01-05:30:00 2014.11.07-17:06:30
edit OTD/test/config/server.xml 2014.11.06-19:48:15 2014.11.07-17:06:08
edit OTD/test/config/test-obj.conf 2014.11.06-16:59:32 1970.01.01-05:29:59
remove OTD/test/config/obj.conf 2014.11.06-19:48:15 1970.01.01-05:30:00
```

showEditSession

Command Category: Editing Commands

Use with WLST: Online

Description

Shows information about the specified edit sessions. If this command is invoked without specifying the name of the edit session, a list of all the existing edit sessions will be displayed. In case the name of a singe edit session is specified as a parameter, detailed information about that edit session will be displayed.

The details about an edit session includes the following information:

- **Creator** - The name of the user who created the edit session
- **Editor** - The name of the user who currently owns the edit session lock
- **Resolve recommended** - If set to yes, this flag indicates that configuration changes in another edit session have been activated and the configuration snapshot in this edit session can be deprecated as such. Use resolve() command to update the content of the session to match the latest runtime configuration or to resolve any modification conflicts or both.
- **Contains unactivated changes** - provides information whether the specified edit session contains any configuration changes that have not been activated yet.

To access detailed information about the default edit session, it is necessary to specify the input parameter as `default`.

Syntax

```
showEditSession([name])
```

Argument	Definition
name	Optional. Name or identifier of the edit session about which to display detailed information. If this argument is not specified, then a list of all existing edit sessions is displayed.

Example

The following example invokes the `showEditSession` command without specifying the argument. In this example, a list of all edit sessions is displayed.

```
wls:/wls/edit> showEditSession()
List of named edit sessions [for details use showEditSession(name)]:
default
one
two
three
```

In the following example, `two` is the name of an edit session which is passed as an argument. As a result, detailed information about the specified edit session is displayed.

```
wls:/wls/edit> showEditSession('two')

two
Creator: wls
Editor (lock owner): wls
Resolve recommended: No
Contains unactivated changes: Yes
```

startEdit

Command Category: Editing Commands

Use with WLST: Online

Description

Starts a configuration edit session on behalf of the currently connected user. You must navigate to the edit configuration MBean hierarchy using the `edit` command before issuing this command. See edit.

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

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

Note:

WLST automatically starts an edit session if it detects that there is an edit session that is already in progress by the same user, which may have been started via the WebLogic Server Administration Console or another WLST session.

Syntax

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

Argument	Definition
`waitTimeInMillis`	Optional. Time (in milliseconds) that WLST waits until it gets a lock, in the event that another user has a lock. This argument defaults to 0 ms.

Argument	Definition
timeOutInMillis	Optional. Timeout (in milliseconds) that WLST waits to release the edit lock. This argument defaults to -1 ms, indicating that this edit session never expires.
exclusive	Optional. Specifies whether the edit session should be an exclusive session. If set to `true`, if the same owner enters the `startEdit` command, WLST waits until the current edit session lock is released before starting the new edit session. The exclusive lock times out according to the time specified in *timeoutInMillis*. This argument defaults to `false`.

Example

The following example starts an edit session with a 1-minute timeout waiting for a lock and a 2-minute timeout waiting to release the edit lock:

```
wls:/mydomain/edit> startEdit(60000, 120000)
Starting an edit session ...
Started edit session, please be sure to save and activate your changes once you
are done.
wls:/mydomain/edit !>
```

stopEdit

Command Category: Editing Commands

Use with WLST: Online

Description

Stops the current edit session, releases the edit lock, and discards unsaved changes.

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

Note:

`stopEdit()` cannot cancel the edit session of a different user, nor can it stop the exclusive edit sessions acquired via other sessions. `stopEdit()` can only stop the non-exclusive edit sessions for the same user.

Syntax

```
stopEdit([defaultAnswer])
```

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

Example

The following example stops the current editing session. WLST prompts for verification before canceling.

```
wls:/mydomain/edit !> stopEdit()
Sure you would like to stop your edit session? (y/n)
y
Edit session has been stopped successfully.
wls:/mydomain/edit>
```

unassign

Command Category: Editing Commands

Use with WLST: Offline

Description

Unassign applications or resources from one or more destinations.

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

Syntax

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

Argument	Definition
`sourceType`	Type of configuration bean to be unassigned. This value can be set to one of the following values: • `AppDeployment` • `Library` • *`securityType`* (such as `User`) • `Server` • *`service`* (such as `JDBCSystemResource`) • *`service`*`.SubDeployment`, where *`service`* specifies the service type of the `SubDeployment` (such as `JMSSystemResource.SubDeployment`); you can also specify nested subdeployments (such as `AppDeployment.SubDeployment.SubDeployment`)
`sourceName`	Name of the application or resource to be unassigned. Multiple names can be specified, separated by commas, or you can use the wildcard (*) character to specify all resources of the specified type. Specify subdeployments using the following format: *`service.subDeployment`*, where *`service`* specifies the parent service and *`subDeployment`* specifies the name of the subdeployment. For example, `myJMSResource.myQueueSubDeployment. You can also specify nested subdeployments, such as MedRecEAR.MedRecAppScopedJMS.MedRecJMSServer.`
`destinationType`	Type of destination. Guidelines for setting this value are provided below.
`destinationName`	Name of the destination. Multiple names can be specified, separated by commas.

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

- When unassigning **application deployments**, set the values as follows:
  - *`sourceType`*: `AppDeployment`
  - *`destinationType`*: `Target`

- When unassigning **libraries**, set the values as follows:
  - *sourceType*: `Library`
  - *destinationType*: `Target`
- When unassigning **security types**, set the values as follows:
  - *sourceType*: Name of the security type, such as `User`
  - *destinationType*: Name of the destination security type, such as `Group`
- When unassigning **servers** from **clusters**, set the values as follows:
  - *sourceType*: `Server`
  - *destinationType*: `Cluster`
- When unassigning **services**, set the values as follows:
  - *sourceType*: Name of the specific server, such as `JDBCSystemResource`
  - *destinationType*: `Target`
- When unassigning **subdeployments**, set the values as follows:
  - *sourceType*: *service*`.SubDeployment`, where *service* specifies the parent of the `SubDeployment`, such as `JMSSystemResource.SubDeployment`; you can also specify nested subdeployments (such as `AppDeployment.SubDeployment.SubDeployment`)
  - *destinationType*: `Target`

Example

The following examples:

- Unassign the servers `myServer` and `myServer2` from the cluster `myCluster`.

```
wls:/offline/medrec> unassign("Server", "myServer,myServer2", "Cluster",
"myCluster")
```

- Unassign all servers from the cluster `myCluster`.

```
wls:/offline/mydomain> unassign("Server", "*", "Cluster", "myCluster")
```

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

```
wls:/offline/medrec> unassign("User", "newUser", "Group", "Monitors")
```

- Unassign the application deployment `myAppDeployment` from the target server `newServer`.

```
wls:/offline/mydomain> unassign("AppDeployment", "myAppDeployment",
"Target", "newServer")
```

- Unassign the nested SubDeployment `MedRecAppScopedJMS.MedRecJMSServer`, which is a child of the AppDeployment `AppDeployment`, from the target server `AdminServer`.

```
wls:/offline/mydomain> assign('AppDeployment.SubDeployment.SubDeployment',
'MedRecEAR.MedRecAppScopedJMS.MedRecJMSServer', 'Target','AdminServer')
```

undo

Command Category: Editing Commands

Use with WLST: Online

Description

Reverts all unsaved or unactivated edits.

You specify whether to revert all unactivated edits (including those that have been saved to disk), or all edits made since the last `save` operation. This command does not release the edit session.

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

Syntax

```
undo([unactivatedChanges], [defaultAnswer])
```

Argument	Definition
`unactivatedChanges`	Optional. Boolean value specifying whether to undo all unactivated changes, including edits that have been saved to disk. This argument defaults to `false`, indicating that all edits since the last `save` operation are reverted.
`defaultAnswer`	Optional. Default response, if you would prefer not to be prompted at the command line. Valid values are `y` and `n`. This argument defaults to null, and WLST prompts you for a response.

Example

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

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

The following example reverts all unactivated changes. WLST prompts for verification before reverting.

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

validate

Command Category: Editing Commands

Use with WLST: Online

Description

Validates the changes that have been made but have not yet been saved. This command enables you to verify that all changes are valid before saving them.

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

Syntax

```
validate()
```

Example

The following example validates all changes that have been made but have not yet been saved.

```
wls:/mydomain/edit !> validate()
Validating changes ...
Validated the changes successfully
```

Information Commands

Use the WLST information commands to interrogate domains, servers, and variables, and provide configuration bean, runtime bean, and WLST-related information. Table 2-8 lists and summarizes these commands.

Table 2-8 Information Commands for WLST Configuration

This command...	Enables you to...	Use with WLST...
addListener	Add a JMX listener to the specified MBean.	Online
configToScript	Convert an existing server configuration (`config` directory) to an executable WLST script	Online or Offline
dumpStack	Display stack trace from the last exception that occurred while performing a WLST action, and reset the stack trace.	Online or Offline
dumpVariables	Display all variables used by WLST, including their name and value.	Online or Offline
find	Find MBeans and attributes in the current hierarchy.	Online
getConfigManager	Return the latest `ConfigurationManagerBean` MBean which manages the change process.	Online
getMBean	Return the MBean by browsing to the specified path.	Online
getMBI	Return the `MBeanInfo` for the specified `MBeanType` or the `cmo` variable.	Online
getPath	Return the MBean path for the specified MBean instance.	Online
listChildTypes	List all the children MBeans that can be created or deleted for the cmo type.	Online
lookup	Look up the specified MBean.	Online
ls	List all child beans and/or attributes for the current configuration or runtime bean.	Online or Offline
man	Display help from `MBeanInfo` for the current MBean or its specified attribute.	Online
redirect	Redirect WLST output to the specified filename.	Online
removeListener	Remove a listener that was previously defined.	Online

Table 2-8 (Cont.) Information Commands for WLST Configuration

This command...	Enables you to...	Use with WLST...
setShowLSResult	Specify whether `ls()` should log its output to stdout.	Online or Offline
showListeners	Show all listeners that are currently defined.	Online
startRecording	Record all user interactions with WLST; useful for capturing commands to replay.	Online or Offline
state	Returns a map of servers or clusters and their state using Node Manager.	Online
stopRecording	Stop recording WLST commands.	Online or Offline
stopRedirect	Stop redirection of WLST output to a file.	Online or Offline
storeUserConfig	Create a user configuration file and an associated key file.	Online
threadDump	Display a thread dump for the specified server.	Online or Offline
viewMBean	Display information about an MBean, such as the attribute names and values, and operations.	Online
writeIniFile	Convert WLST definitions and method declarations to a Python (`.py`) file.	Online or Offline

addListener

Command Category: Information Commands

Use with WLST: Online

Description

Adds a JMX listener to the specified MBean. Any changes made to the MBean are reported to standard out and/or are saved to the specified configuration file.

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

Syntax

```
addListener(mbean, [attributeNames], [logFile], [listenerName])
```

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

Argument	Definition
listenerName	Optional. Name of the JMX listener. This argument defaults to a WLST-generated name.

Example

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

```
wls:/mydomain/serverConfig> addListener(cmo, "Notes,ArchiveConfigurationCount",
"./listeners/domain.log","domain-listener")
```

configToScript

Command Category: Information Commands

Use with WLST: Online or Offline

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

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.

The `configToScript` command is deprecated and will be removed in a future release of WebLogic Server. Oracle strongly recommends that you use `pack` and `unpack` to create the domain on remote servers. See Overview of the Pack and Unpack Commands in *Creating Templates and Domains Using the Pack and Unpack Commands*.

The `configToScript` command creates the following files:

- A WLST script that contains the commands needed to recreate the configuration.
- A properties file that contains domain-specific values. You can update the values in this file to create new domains that are similar to the original configuration.
- A user configuration file and an associated key file to store encrypted attributes. The user configuration file contains the encrypted information. The key file contains a secret key that is used to encrypt and decrypt the encrypted information.

When you run the generated script:

- If a server is currently running, WLST will try to connect using the values in the properties file and then run the script commands to create the server resources.

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

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

Syntax

```
configToScript([configPath], [pyPath], [overwrite], [propertiesFile],
[createDeploymentScript])
```

Argument	Definition
`configPath`	Optional. Path to the `domain` directory that contains the configuration that you want to convert. This argument defaults to the directory from which you start WLST `(./)`.
`pyPath`	Optional. Path and filename to which you want to write the converted WLST script. This argument defaults to `./config/config.py`.
`overwrite`	Optional. Boolean value specifying whether the script file should be overwritten if it already exists. This argument defaults to `true`, indicating that the script file is overwritten.
`propertiesFile`	Optional. Path to the directory in which you want WLST to write the properties files. This argument defaults to the pathname specified for the `scriptPath` argument.
`createDeploymentScript`	Optional. Boolean value specifying whether WLST creates a script that performs deployments only. This argument defaults to `false`, indicating that a deployment script is not created.

Example

The following example converts the configuration to a WLST script `config.py`. By default, the configuration file is loaded from `./config`, the script file is saved to `.config/config.py`, and the properties files is saved to `.config/config.py.properties`.

```
wls:/offline> configToScript()
configToScript is loading configuration from c:\Oracle\Middleware
\user_projects\domains\wls\config\config.xml ...
Completed configuration load, now converting resources to wlst script...
configToScript completed successfully
The WLST script is written to c:\Oracle\Middleware
\user_projects\domains\wls\config\config.py
and the properties file associated with this script is written to c:\Oracle\
Middleware\user_projects\domains\wls\config\config.py.properties
wls:/offline>
```

The following example converts server resources configured in the file `c:\Oracle\Middleware\user_projects\domains\mydomain\config` directory to a WLST script `c:\Oracle\Middleware\myscripts\config.py`.

```
wls:/offline> configToScript('c:/Oracle/Middleware/user_projects/domains
/mydomain','c:/Oracle/Middleware/myscripts')
configToScript is loading configuration from c:\Oracle\Middleware
\user_projects\domains\mydomain\config\config.xml ...
Completed configuration load, now converting resources to wlst script...
configToScript completed successfully
The WLST script is written to c:\Oracle\Middleware\myscripts\config.py
and the properties file associated with this script is written to
```

```
c:\Oracle\Middlware\mydomain\config.py.properties
wls:/offline>
```

dumpStack

Command Category: Information Commands

Use with WLST: Online or Offline

Description

Displays the stack trace from the last exception that occurred while performing a WLST action, and resets the stack trace.

If successful, the `dumpStack` command prints the stack trace from the Throwable object, or in the event of an error, from the `WLSTException`.

Syntax

```
dumpStack()
```

Example

This example displays the stack trace.

```
wls:/myserver/serverConfig> dumpStack()
com.bea.plateng.domain.script.jython.WLSTException: java.lang.reflect.Invocation
TargetException
...
```

dumpVariables

Command Category: Information Commands

Use with WLST: Online or Offline

Description

Displays all the variables used by WLST, including their name and value. In the event of an error, the command returns a `WLSTException`.

Syntax

```
dumpVariables()
```

Example

This example displays all the current variables and their values.

```
wls:/mydomain/serverConfig> dumpVariables()
adminHome   weblogic.rmi.internal.BasicRemoteRef - hostID:
   '-1 108080150904263937S:localhost:[7001,8001,-1,-1,-1,-1,-1]:
   mydomain:AdminServer', oid: '259', channel: 'null'
cmgr   [MBeanServerInvocationHandler]com.bea:Name=ConfigurationManager,
   Type=weblogic.management.mbeanservers.edit.ConfigurationManagerMBean
cmo   [MBeanServerInvocationHandler]com.bea:Name=mydomain,Type=Domain
connected true
domainName mydomain
...
wls:/mydomain/serverConfig>
```

find

Command Category: Information Commands

Use with WLST: Online

Description

Finds MBeans and attributes in the current hierarchy.

WLST returns the pathname to the MBean that stores the attribute and/or attribute type, and its value. If `searchInstancesOnly` is set to `false`, this command also searches the MBeanType paths that are not instantiated in the server, but that can be created. In the event of an error, the command returns a `WLSTException`.

Syntax

```
find([name], [type], [searchInstancesOnly])
```

Argument	Definition
`name`	Optional. Name of the attribute to find.
`type`	Optional. Type of the attribute to find.
`searchInstancesOnly`	Optional. Boolean value specifying whether to search registered instances only or to also search `MBeanTypes` paths that are not instantiated in the server, but that can be created. This argument defaults to `true`, indicating only the registered instances will be searched.

Example

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

```
wls:/mydomain/serverConfig> find(name = 'JavaCompiler')
Finding 'JavaCompiler' in all registered MBean instances ...
/Servers/AdminServer                         JavaCompilerPreClassPath      null
/Servers/AdminServer                        JavaCompiler                  java
/Servers/AdminServer                       JavaCompilerPostClassPath   null
wls:/mydomain/serverConfig>
```

The following example searches for an attribute of type `JMSRuntime` in the current configuration hierarchy.

```
wls:/mydomain/serverRuntime> find(type='JMSRuntime')
Finding MBean of type 'JMSRuntime' in all the instances ...
/JMSRuntime/AdminServer.jms
wls:/mydomain/serverRuntime>
```

The following example searches for an attribute named `execute` in the current configuration hierarchy. The `searchInstancesOnly` argument is set to `false`, indicating to also search MBeanTypes that are not instantiated in the server.

```
wls:/mydomain/serverConfig> find(name='execute', searchInstancesOnly='false')
Finding 'execute' in all registered MBean instances ...
/Servers/AdminServer      ExecuteQueues [Ljavax.management.ObjectName;@1aa7dbc
/Servers/AdminServer       Use81StyleExecuteQueues                       false
Now finding 'execute' in all MBean Types that can be instantiated ...
```

```
/Servers                                              ExecuteQueues
/Servers                                              Use81StyleExecuteQueues
wls:/mydomain/serverConfig>
```

getConfigManager

Command Category: Information Commands

Use with WLST: Online

Description

Returns the latest `ConfigurationManager` MBean, which manages the change process. You can then invoke methods to manage configuration changes across a WebLogic domain. In the event of an error, the command returns a `WLSTException`.

Syntax

```
getConfigManager()
```

Example

The following example returns the latest `ConfigurationManagerBean` MBean and stores it in a `cm` variable.

```
wls:/mydomain/serverConfig> cm=getConfigManager()
wls:/mydomain/serverConfig> cm.getType()
'weblogic.management.mbeanservers.edit.ConfigurationManagerMBean'
```

getMBean

Command Category: Information Commands

Use with WLST: Online

Description

Returns an MBean by browsing to the specified path. In the event of an error, the command returns a `WLSTException`.

Note:

No exception is thrown if the MBean is not found.

Syntax

```
getMBean(mbeanPath)
```

Argument	Definition
mbeanPath	Path name to the MBean in the current hierarchy.

Example

The following example returns the MBean specified by the path.

```
wls:/mydomain/edit !> com=getMBean('Servers/myserver/COM/myserver')
wls:/mydomain/edit !> com.getType()
'Server'
```

getMBI

Command Category: Information Commands

Use with WLST: Online

Description

Returns the `MBeanInfo` for the specified `MBeanType` or the `cmo` variable. In the event of an error, the command returns a `WLSTException`.

Syntax

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

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

Example

The following example gets the `MBeanInfo` for the specified `MBeanType` and stores it in the variable `svrMbi`.

```
wls:/mydomain/serverConfig>
svrMbi=getMBI('weblogic.management.configuration.ServerMBean')
```

getPath

Command Category: Information Commands

Use with WLST: Online

Description

Returns the MBean path for the specified MBean instance or ObjectName for the MBean in the current tree. In the event of an error, the command returns a `WLSTException`.

Syntax

```
getPath(mbean)
```

Argument	Definition
`mbean`	MBean instance or ObjectName for the MBean in the current tree for which you want to return the MBean path.

Example

The following example returns the MBean specified by the path.

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

listChildTypes

Command Category: Information Commands

Use with WLST: Online

Description

Lists all of the child MBeans that can be created or deleted for the `cmo`. The `cmo` variable specifies the configuration bean instance to which you last navigated using WLST. See Changing the Current Management Object in *Understanding the WebLogic Scripting Tool*.

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

Syntax

```
listChildTypes([parent])
```

Argument	Definition
parent	Optional. Parent type for which you want the children types listed.

Example

The following example lists the children MBeans that can be created or deleted for the `cmo` type.

```
wls:/mydomain/serverConfig> listChildTypes()
AppDeployments
BridgeDestinations
CachingRealms
Clusters
...
wls:/mydomain/serverConfig>
```

lookup

Command Category: Information Commands

Use with WLST: Online

Description

Looks up the specified MBean. The MBean must be a child of the current MBean. In the event of an error, the command returns a `WLSTException`.

Syntax

```
lookup(name, [childMBeanType])
```

Argument	Definition
name	Name of the MBean that you want to lookup.
childMBeanType	Optional. The type of the MBean that you want to lookup.

Example

The following example looks up the specified server, `myserver`, and stores the returned stub in the `sbean` variable.

```
wls:/mydomain/serverConfig> sbean=lookup('myserver','Server')
wls:/mydomain/serverConfig> sbean.getType()
'Server'
wls:/mydomain/serverConfig>
```

ls

Command Category: Information Commands

Use with WLST: Online or Offline

Description

Lists all of the child beans and/or attributes for the current configuration or runtime bean. You can optionally control the output by specifying an argument. If no argument is specified, the command lists all of the child beans and attributes in the domain. The output is returned as a string.

Note:

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

Because WLST online caches child beans for performance reasons, an `ls()` may not return child beans that were created in another process after the initial `ls()` was issued for the current bean. If a child bean is not present, then `cd` to the parent directory and then back to refresh the cache.

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

By default, the output is returned as a string and is arranged in three columns.

Note:

By default, the `ls()` function echoes its output so that it appears on the console in which you are running WLST. You can disable this by including the following command in your WLST script:

```
WLS.setShowLSResult(flag)
```

`flag` is a Boolean value. If it is `0` (false), then output from `ls()` to stdout is disabled and will not appear on the console. If it is set to `1` (true), then output from `ls()` to stdout is enabled.

- The first column displays a set of codes that describe the listed item. See Table 2-9.
- The second column displays the item name.
- When the item is an attribute, the third column displays the attribute value. If an attribute is encrypted, the third column displays asterisks instead of the value. (See Writing and Reading Encrypted Configuration Values in *Understanding the WebLogic Scripting Tool*.)
- When the item is an operation, the third column uses the following pattern to display the operation's return type and input parameters: `returnType: parameterType(parameterName)`

Table 2-9 ls Command Output Information

Code	Description
`d`	Indicates that the item is a child management object. As with a directory in a UNIX or Windows file system, you can use the `cd` command to make the child object the current management object.
`r`	Indicates that the item is a child management object or an attribute that is readable, assuming that current user has been given read permission by the security realm's policies. (See Default Security Policies for MBeans in the *MBean Reference for Oracle WebLogic Server*.)
`w`	Indicates that the item is an attribute that is writable, assuming that current user has been given write permission by the security realm's policies. (See Default Security Policies for MBeans in the *Oracle WebLogic Server MBean Reference*.)
`x`	Indicates that the item is an operation that can be executed, assuming that current user has been given execute permission by the security realm's policies. (See Default Security Policies for MBeans in the *MBean Reference for Oracle WebLogic Server*.)

By default, the output lists all attributes, operations, and child management objects of the current management object. To filter the output or to see a list for a different management object, you can specify a command argument.

Note:

As a performance optimization, when using WLST offline, WebLogic Server does not store most of its default values in the configuration files for the WebLogic domain. 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 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 `Log` management object for the domain.

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.

Syntax

```
ls( [ a | c | o ] [ moPath ])

ls( [ moPath ] [returnMap] [ returnType ] [inheritance])
```

Argument	Definition
`a`	Optional. Displays only the attributes of the specified management object (suppresses the display of other items).
`c`	Optional. Displays only the child management objects of the specified management object (suppresses the display of other items).
`o`	Optional. Displays only the operations that can be invoked on the specified management object (suppresses the display of other items). This argument is only applicable for WLST online.
`moPath`	Optional. Path name to the management object for which you want to list attributes, operations, and child management objects. You can specify a pathname that is relative to your current location in the hierarchy or an absolute pathname. With WLST offline, use the forward-slash character (`/`) to specify the root of the configuration document. With WLST online, you can list the contents of MBeans in any management hierarchy (see Tree Commands). Use the following syntax to specify the root of a hierarchy: *`root-name`*`:/` For example, to list the root of the server runtime hierarchy: `ls('serverRuntime:/')` If you do not specify this argument, the command lists items for the current management object.
`returnMap`	Optional. Boolean value that determines whether the command returns values as a map. This argument defaults to `false`, which causes this command to return a String.
`returnType`	Optional. Controls the output returned in the map. Specify `a`, `c`, or `o`, which filter the output as described at the top of this table. This argument is valid only if `returnMap` is set to `true`. This argument defaults to `c`.
`inheritance`	Optional. Boolean value that specifies whether to include inheritance information for each attribute value. If specified, then WLST will identify attribute values that are inherited from a server template by appending the value with an (Inherited) suffix. This argument defaults to `false`. It can be used only in WLST online mode. For information about server templates, see Server Templates in *Understanding Domain Configuration for Oracle WebLogic Server*.

Example

The following example displays all of the child configuration beans, and attribute names and values for the `examples` domain, which has been loaded into memory, in WLST offline mode:

```
wls:/offline/mydomain > ls()
dr--   AppDeployments
dr--   BridgeDestinations
dr--   Clusters
```

```
dr--   CustomResources
dr--   DeploymentConfiguration
dr--   Deployments
dr--   EmbeddedLDAP
dr--   ErrorHandlings
dr--   FileStores
dr--   InternalAppDeployments
dr--   InternalLibraries
dr--   JDBCDataSourceFactories
dr--   JDBCStores
dr--   JDBCSystemResources
dr--   JMSBridgeDestinations
dr--   JMSInteropModules
dr--   JMSServers
dr--   JMSSystemResources
dr--   JMX
...
wls:/offline/examples>
```

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

```
wls:/mydomain/serverConfig> ls('a')
-r--   AdminServerName                              AdminServer
-r--   AdministrationMBeanAuditingEnabled           false
-r--   AdministrationPort                           9002
-r--   AdministrationPortEnabled                    false
-r--   AdministrationProtocol                       t3s
-r--   ArchiveConfigurationCount                    0
-r--   ClusterConstraintsEnabled                    false
-r--   ConfigBackupEnabled                          false
-r--   ConfigurationAuditType                       none
-r--   ConfigurationVersion                         9.0.0.0
-r--   ConsoleContextPath                           console
-r--   ConsoleEnabled                               true
-r--   ConsoleExtensionDirectory                    console-ext
-r--   DomainVersion                                9.0.0.0
-r--   LastModificationTime                         0
-r--   Name                                         basicWLSDomain
-r--   Notes                                        null
-r--   Parent                                       null
-r--   ProductionModeEnabled                        false
-r--   RootDirectory                                .
-r--   Type                                         Domain
wls:/mydomain/serverConfig>
```

The following example displays all of the child beans, and attribute names and values in `Servers` MBean:

```
wls:/mydomain/serverConfig> ls('Servers')
dr--   AdminServer
```

The following example displays the attribute names and values for the specified MBean path and returns the information in a map:

```
wls:/mydomain/serverConfig> svrAttrList = ls('edit:/Servers/myserver', 'true',
'a')
-rw-   AcceptBacklog                                50
-rw-   AdminReconnectIntervalSeconds                10
-rw-   AdministrationPort                           9002
-rw-   AdministrationProtocol                       t3s
-rw-   AutoKillIfFailed                             false
```

```
-rw-   AutoMigrationEnabled                          false
-rw-   AutoRestart                                   true
-rw-   COMEnabled                                    false
-rw-   ClasspathServletDisabled                      false
-rw-   ClientCertProxyEnabled                        false
-rw-   Cluster                                       null
-rw-   ClusterRuntime                                null
-rw-   ClusterWeight                                 100
wls:/mydomain/serverConfig>
```

The following command shows the output of `ls()` for the server `server1`, which inherits the `RestartIntervalSeconds` and `RestartMax` attributes from a defined server template:

```
wls:/domain1/serverConfig/Servers> ls('new_ManagedServer_1','true','a','true')

-r--   AcceptBacklog                                 300
.
.
.
-r--   RestartIntervalSeconds (Inherited)            3000
-r--   RestartMax (Inherited)                           5
.
.
.
-r--   XMLRegistry                                   null

wls:/domain1/serverConfig/Servers>
```

man

Command Category: Information Commands

Use with WLST: Online

Description

Displays help from `MBeanInfo` for the current MBean or its specified attribute. In the event of an error, the command returns a `WLSTException`.

Syntax

```
man([a | c | o | attrName])
```

Argument	Definition
`a`	Optional. Displays help for all the attributes for the current MBean.
`c`	Optional. Displays help for all the child configuration beans that are contained in the current MBean.
`o`	Optional. Displays help for all operations that can be invoked on the current MBean.
`attrName`	Optional. MBean attribute name for which you would like to display help. If not specified, WLST displays help for the current MBean.

Example

The following example displays help from `MBeanInfo` for the `ServerMBean` bean.

```
wls:/mydomain/serverConfig> man('Servers')
dynamic : true
```

```
creator : createServer
destroyer : destroyServer
description : <p>Returns the ServerMBeans representing the servers that have
been
configured to be part of this domain.</p>
descriptorType : Attribute
Name : Servers
interfaceClassName : [Lweblogic.management.configuration.ServerMBean;
displayName : Servers
relationship : containment
```

redirect

Command Category: Information Commands

Use with WLST: Online

Description

Redirects WLST information, error, and debug messages to the specified filename. Also redirects the output of the `dumpStack()` and `dumpVariables()` commands to the specified filename.

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

Syntax

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

Argument	Definition
`outputFile`	Name of the file to which you want to record the WLST commands. The filename can be absolute or relative to the directory from which you started WLST.
`toStdOut`	Optional. Boolean value specifying whether the output should be sent to `stdout`. This argument defaults to `true`, indicating that the output will be sent to `stdout`.

Example

The following example begins redirecting WLST output to the `logs/wlst.log` file:

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

removeListener

Command Category: Information Commands

Use with WLST: Online

Description

Removes a previously defined listener. If you do not specify an argument, WLST removes all listeners defined for all MBeans. For information about setting a listener, see addListener.

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

Syntax

```
removeListener([mbean], [listenerName])
```

Argument	Definition
mbean	Optional. Name of the MBean or MBean object for which you want to remove the previously defined listeners.
listenerName	Optional. Name of the listener to be removed.

Example

The following example removes the listener named `mylistener`.

```
wls:/mydomain/serverConfig> removeListener(listenerName="mylistener")
```

setShowLSResult

Command Category: Information Commands

Use with WLST: Online or Offline

Description

Set this command to specify whether the `ls()` function should log its output to stdout or console. Set value to `false` to disable the output from `ls()` to stdout. Set value to `true` to enable the output from `ls()` to stdout.

Syntax

```
setShowLSResult()
```

Argument	Definition
showResult	A Boolean value. If it is set to `false`, then the output from `ls()` to stdout is disabled. If set to `true`, then output from `ls()` to stdout is enabled.

Example

The following example disables the output from `ls()` to stdout in offline mode.

```
wls:/offline/base_domain> setShowLSResult(false)
wls:/offline/base_domain> ls()
```

The following example enables ls() to print output to stdout in online mode.

```
wls:/base_domain/serverConfig> setShowLSResult(true)
wls:/base_domain/serverConfig> ls()
dr--   AdminConsole
```

showListeners

Command Category: Information Commands

Use with WLST: Online

Description

Shows all currently defined listeners. For information about setting a listener, see addListener.

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

Syntax

```
showListeners()
```

Example

The following example shows all listeners that are currently defined.

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

startRecording

Command Category: Information Commands

Use with WLST: Online or Offline

Description

Records all user interactions with WLST. This command is useful for capturing commands for replay.

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

This command cannot be used when you are importing WLST as a Jython module, as described in Importing WLST as a Jython Module in *Understanding the WebLogic Scripting Tool*.

Syntax

```
startRecording(recordFile, [recordAll])
```

Argument	Definition
recordFile	Name of the file to which you want to record the WLST commands. The filename can be absolute or relative to the directory from which you invoked WLST.
recordAll	Optional. Boolean value specifying whether to capture all user interactions in the file. This argument defaults to `false`, indicating that only WLST commands are captured, and not WLST command output.

Example

The following example begins recording WLST commands in the `record.py` file:

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

state

Command Category: Information Commands

Use with WLST: Online

Description

Uses Node Manager to return a map of servers, clusters, or system components and their state. Node Manager must be running.

See Understanding Server Life Cycle in *Administering Server Startup and Shutdown for Oracle WebLogic Server*.

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

Syntax

```
state(name, [type], [returnMap])
```

Argument	Definition
name	Name of the server, cluster, or system component for which you want to retrieve the current state.
type	Optional. Type is `Server`, `Cluster`, or `SystemComponent`. If not specified, WLST will look for a server, cluster, or system component with the specified name.
returnMap	Optional. Boolean value that specifies whether or not to return a map containing the return values. If `returnMap` is `true`, the `state` command returns a map of all states. The default is `false`.

Example

The following example returns the state of the Managed Server, `managed1`.

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

The following example returns the state of the cluster, `mycluster`. The optional `type` argument is excluded.

```
wls:/mydomain/serverConfig> state('mycluster')
There are 3 server(s) in cluster: mycluster

States of the servers are
MServer1---SHUTDOWN
MServer2---SHUTDOWN
MServer3---SHUTDOWN
wls:/mydomain/serverConfig>
```

The following example shows the return map that is printed to standard output when returnMap is set to `false`.

```
wls:/mydomain/serverConfig> retMap=state('managed1')
Current state of "managed1": SUSPENDED
wls:/mydomain/serverConfig> print retMap
None
```

The following example shows the return map that is printed to standard output when `returnMap` is set to `true`.

```
wls:/mydomain/serverConfig> retMap=state('managed1', returnMap='true')
Current state of "managed1": SUSPENDED
wls:/mydomain/serverConfig> print retMap
{managed1=SUSPENDED}
```

stopRecording

Command Category: Information Commands

Use with WLST: Online or Offline

Description

Stops recording WLST commands. For information about starting a recording, see startRecording.

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

Syntax

```
stopRecording()
```

Example

The following example stops recording WLST commands.

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

stopRedirect

Command Category: Information Commands

Use with WLST: Online or Offline

Description

If redirection is in progress, stops the redirection of WLST output to a file.

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

Syntax

```
stopRedirect()
```

Example

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

```
wls:/mydomain/serverConfig> stopRedirect()
WLST output will not be redirected to myfile.txt any more
```

storeUserConfig

Command Category: Information Commands

Use with WLST: Online

Description

Creates a user configuration file and an associated key file. The user configuration file contains an encrypted username and password. The key file contains a secret key that is used to encrypt and decrypt the username and password.

Only the key file that originally encrypted the username and password can be used to decrypt the values. If you lose the key file, you must create a new user configuration and key file pair.

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

Syntax

```
storeUserConfig([userConfigFile], [userKeyFile], [nm])
```

Argument	Definition
`userConfigFile`	Optional. Name of the file to store the user configuration. The pathname can be absolute or relative to the file-system directory from which you started WLST. If you do not specify this argument, the command stores the file in your home directory as determined by your JVM. The location of the home directory depends on the SDK and type of operating system on which WLST is running. The default filename is based on the following pattern, where *`username`* is the user name that you used to log in to the operating system. *`username`*`-WebLogicConfig.properties` The command also prints to standard out the location in which it created the file.
`userKeyFile`	Optional. Name of the file to store the key information that is associated with the user configuration file that you specify. The pathname can be absolute or relative to the file-system directory from which you started WLST. If you do not specify this argument, the command stores the file in your home directory as determined by your JVM. The location of the home directory depends on the SDK and type of operating system on which WLST is running. The default filename is based on the following pattern, where *`username`* is the user name that you used to log in to the operating system. *`username`*`-WebLogicKey.properties` The command also prints to standard out the location in which it created the file.
`nm`	Optional. Boolean value that specifies whether to store the username and password for Node Manager. If set to true, the Node Manager username and password is stored. This argument defaults to `false`.

Example

The following example creates and stores a user configuration file and key file in the default location.

```
wls:/mydomain/serverConfig>  storeUserConfig()
Creating the key file can reduce the security of your system if it is not kept in
a secured location after it is created. Do you want to create the key file? y or n
y
The username and password that were used for this current WLS connection are
stored in C:\Documents and Settings\pat\pat-WebLogicConfig.properties
and C:\Documents and Settings\pat\pat-WebLogicKey.properties.
```

The following example creates and stores a user configuration file and key file in the specified locations.

```
wls:/mydomain/serverConfig>  storeUserConfig('c:/myFiles/
myuserconfigfile.secure', 'c:/myFiles/myuserkeyfile.secure')
Creating the key file can reduce the security of your system if it is not kept
in
a secured location after it is created. Do you want to create the key file? y or
n
y
The username and password that were used for this current WLS connection are
stored in c:/myFiles/mysuserconfigfile.secure and c:/myFiles/myuserkeyfile.secure
wls:/mydomain/serverConfig>
```

threadDump

Command Category: Information Commands

Use with WLST: Online or Offline

Description

Displays a thread dump for the specified server. In the event of an error, the command returns a `WLSTException`.

Syntax

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

Argument	Definition
writeToFile	Optional. Boolean value specifying whether to save the output to a file. This argument defaults to `true`, indicating that output is saved to a file.
fileName	Optional. Name of the file to which the output is written. The filename can be absolute or relative to the directory where WLST is running. This argument defaults to `Thread_Dump_`*serverName* file, where *serverName* indicates the name of the server. This argument is valid only if *writeToFile* is set to `true`.
serverName	Optional. Server name for which the thread dump is requested. This argument defaults to the server to which WLST is connected. If you are connected to an Administration Server, you can display a thread dump for the Administration Server and any Managed Server that is running in the WebLogic domain. If you are connected to a Managed Server, you can only display a thread dump for that Managed Server.

Example

The following example displays the thread dump for the current server and saves the output to the `Thread_Dump_`*serverName* file.

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

The following example displays the thread dump for the server `managedServer`. The information is not saved to a file.

```
wls:/mydomain/serverConfig> threadDump(writeToFile='false',
serverName='managedServer')
```

viewMBean

Command Category: Information Commands

Use with WLST: Online

Description

Displays information about an MBean, such as the attribute names and values, and operations. In the event of an error, the command returns a `WLSTException`.

Syntax

```
viewMBean(mbean)
```

Argument	Definition
mbean	MBean for which you want to display information.

Example

The following example displays information about the current MBean, `cmo`.

```
wls:/mydomain/serverConfig> cmo.getType()
'Domain'
wls:/mydomain/serverConfig> viewMBean(cmo)
Attribute Names and Values
--------------------------
XMLEntityCaches   null
Targets   javax.management.ObjectName[com.bea
:Name=MedRecJMSServer,Type=JMSServer,
   com.bea:Name=WSStoreForwardInternalJMSServerMedRecServer,Type=JMSServer,
   com.bea:Name=MedRecWseeJMSServer,Type=JMSServer,
   com.bea:Name=PhysWSEEJMSServer,Type=JMSServer,
   com.bea:Name=MedRecSAFAgent,Type=SAFAgent,
   com.bea:Name=AdminServer,Type=Server]
RootDirectory                                  .
EmbeddedLDAP                         com.bea:Name=OOTB_medrec,Type=EmbeddedLDAP
RemoteSAFContexts  null
Libraries   javax.management.ObjectName[com.bea
...
wls:/mydomain/serverConfig>
```

writeIniFile

Command Category: Information Commands

Use with WLST: Online

Description

Converts WLST definitions and method declarations to a Python (`.py`) file to enable advanced users to import them as a Jython module. After importing, the definitions and method declarations are available to other Jython modules and can be accessed directly using Jython syntax. See Importing WLST as a Jython Module in *Understanding the WebLogic Scripting Tool*.

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

Syntax

```
writeIniFile(filePath)
```

Argument	Definition
filePath	Full pathname to the file that you want to save the converted information.

Example

The following example converts WLST to a Python file named `wl.py`.

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

Life Cycle Commands

Use the WLST life cycle commands to manage the life cycle of a server instance.
Table 2-10 lists and summarizes these commands.
See Understanding Server Life Cycle in *Administering Server Startup and Shutdown for Oracle WebLogic Server*.

Table 2-10 Life Cycle Commands for WLST Configuration

This command...	Enables you to...	Use with WLST...
migrate	Migrate services to a target server within a cluster.	Online
resume	Resume a server instance that is suspended or in `ADMIN` state.	Online
scaleDown	Decrease the number of running dynamic servers in the specified dynamic cluster.	Online
scaleUp	Increase the number of running dynamic servers for the specified dynamic cluster.	Online
shutdown	Gracefully shut down a running server instance, cluster, or system component.	Online
softRestart	Restart a system component server instance.	Online
start	Start a Managed Server instance or a cluster using Node Manager.	Online
startServer	Start the Administration Server.	Online or Offline
suspend	Suspend a running server.	Online

migrate

Command Category: Life Cycle Commands

Use with WLST: Online

Description

Migrates the specified services (JTA, JMS, or Server) to a targeted server within a cluster. In the event of an error, the command returns a `WLSTException`.

For information about migrating services, see Service Migration in *Administering Clusters for Oracle WebLogic Server*.

Syntax

```
migrate(sname, destinationName, [sourceDown], [destinationDown], [migrationType])
```

Argument	Definition
sname	Name of the Managed Server where currently the migratable services are hosted. Also known as the Current Hosting Server.
destinationName	Name of the machine or server to which you want to migrate the services.
sourceDown	Optional. Boolean value specifying whether the source server is down. This argument defaults to `true`, indicating that the source server is not running. When migrating JTA services, the *sourceDown* argument is ignored, if specified, and defaults to `true`. The source server **must** be down in order for the migration of JTA services to succeed.
destinationDown	Optional. Boolean value specifying whether the destination server is down. This argument defaults to `false`, indicating that the destination server is running. If the destination is not running, and you do not set this argument to `true`, WLST returns a `MigrationException`. When migrating JMS-related services to a non-running server instance, the server instance will activate the JMS services upon the next startup. When migrating the JTA Transaction Recovery Service to a non-running server instance, the target server instance will assume recovery services when it is started.
migrationType	Optional. Type of service(s) that you want to migrate. Valid values include: • `jms`—Migrate JMS-related services (JMS server, SAF agent, path service, and the WebLogic persistent store) only. • `jta`—Migrate JTA services only. • `server`—Migrate Server services only. • `all`—Migrate all JTA and JMS services. This argument defaults to `all`.

Example

The following example migrates all JMS and JTA services on `server1` to the server `server2`. The boolean arguments specify that the source server is down and the destination server is running.

```
wls:/mydomain/edit !> migrate('server1','server2', 'true', 'false', 'all')
Migrating all JMS and JTA services from 'server1' to destination 'server2' ...
wls:/mydomain/edit !>
```

The following example migrates all Server services on `server1` to the server `server2`. The boolean arguments specify that the source server is down and the destination server is running.

```
wls:/mydomain/edit !> migrate('server1','server2', 'true', 'false', 'Server')
Migrating singleton server services from 'server1' to machine 'server2'...
wls:/mydomain/edit !>
```

resume

Command Category: Life Cycle Commands

Use with WLST: Online

Description

Resumes a server instance that is suspended or in `ADMIN` state. This command moves a server to the `RUNNING` state. See Understanding Server Life Cycle in *Administering Server Startup and Shutdown for Oracle WebLogic Server*.

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

Syntax

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

Argument	Definition
`sname`	Name of the server to resume. This argument defaults to the server to which WLST is currently connected.
`block`	Optional. Boolean value specifying whether WLST should block user interaction until the server is resumed. This argument defaults to `false`, indicating that user interaction is not blocked. In this case, WLST returns control to the user after issuing the command and assigns the task MBean associated with the current task to a variable that you can use to check its status. If you are importing WLST as a Jython module, as described in Importing WLST as a Jython Module in *Understanding the WebLogic Scripting Tool*, *`block`* is always set to `true`.

Example

The following example resumes a Managed Server instance.

```
wls:/mydomain/serverConfig> resume('managed1', block='true')
Server 'managed1' resumed successfully.
wls:/mydomain/serverConfig>
```

scaleDown

Command Category: Life Cycle Commands

Use with WLST: Online

Description

Gracefully shuts down the specified number of running dynamic servers for the specified dynamic cluster. The server with the highest server ID will be shut down first, followed by the server with the next highest server ID, until the specified number of servers have been shut down. The user may, optionally, specify to decrease the size of the dynamic cluster.

Note:

The `scaleDown` command lowers the cluster dynamic cluster size setting when its `updateConfiguration` option is enabled, which in turn can abandon JMS persistent messages and JTA transactions that are associated with retired servers. If you are using JMS or JTA, do not use `scaleDown` with its `updateConfiguration` option enabled. See Best Practices for Using Cluster Targeted JMS Services in *Administering JMS Resources for Oracle WebLogic Server*.

Syntax

```
scaleDown (clusterName, numServers, [updateConfiguration], [block], [timeoutSeconds],
[type])
```

Argument	Definition
`clusterName`	Name of the dynamic cluster.
`numServers`	The number of servers to shut down.
`updateConfiguration`	Optional. Boolean value specifying whether WLST should decrease the size of the dynamic cluster by updating the value of the `DynamicServersMBean.DynamicClusterSize` attribute.
`block`	Optional. Boolean value specifying whether WLST should block user interaction until the system component is started. This argument defaults to `true`, indicating that user interaction is blocked until the operation completes. If set to `false`, WLST returns control to the user after issuing the command and assigns the task MBean associated with the current task to a variable that you can use to check its status. If you are importing WLST as a Jython module, as described in Importing WLST as a Jython Module in *Understanding the WebLogic Scripting Tool*, `block` is always set to `true`.
`timeoutSeconds`	Optional. Time (in seconds) that WLST waits for the server(s) to shut down before canceling the operation. The default value is 300 seconds.
`type`	Optional. If specified, the argument value must be `DynamicCluster`.

Example

The following example scales down the dynamic cluster, `myCluster` by two servers, and blocks the user interaction until the operation completes.

```
wls:/myDomain/serverConfig> scaleDown('myCluster', 2, true, true)
            Remote Scaledown started successfully after 0 seconds.
            Waiting for 2 servers to stop. The timeout is 300 seconds.

            The servers were stopped successfully.
            wls:/myDomain/serverConfig>
```

scaleUp

Command Category: Life Cycle Commands

Use with WLST: Online

Description

Increases the number of running dynamic servers for the specified dynamic cluster. The non-running server with the lowest server ID starts first, followed by the non-running server with the next highest server ID. The user may, optionally, increase the size of the dynamic cluster configuration if there are not enough non-running servers in the cluster to start. If the `updateConfiguration` argument is specified, then the dynamic cluster size setting of the cluster is increased by the additional number of servers, and the specified number of servers are started.

Syntax

```
scaleUp (clusterName, numServers, [updateConfiguration], [block],
[timeoutSeconds], [type])
```

Argument	Definition
clusterName	Name of the dynamic cluster.
numServers	The number of servers to be started.
updateConfiguration	Optional. Boolean value specifying whether WLST should increase the size of the dynamic cluster if there are not enough non-running servers. If not specified, this argument defaults to `false`.
block	Optional. Boolean value specifying whether WLST should block user interaction until the system component is started. This argument defaults to `true`, indicating that user interaction is blocked until the operation completes. If set to `false`, WLST returns control to the user after issuing the command and assigns the task MBean associated with the current task to a variable that you can use to check its status. If you are importing WLST as a Jython module, as described in Importing WLST as a Jython Module in *Understanding the WebLogic Scripting Tool*, `block` is always set to `true`.
timeoutSeconds	Optional. Time (in seconds) that WLST waits for the servers to start before canceling the operation. The default value is 600 seconds.
type	Optional. If specified, the argument value must be `DynamicCluster`.

Example

The following example scales up the dynamic cluster, `myCluster` by specifying two additional servers to be started, and blocking the user interaction until the operation completes.

```
wls:/myDomain/serverConfig> scaleUp('myCluster', 2, true, true)
            Remote Scaleup started successfully after 0 seconds.
            Waiting for 2 servers to reach the running state.
            The timeout is 600 seconds.

            1 server(s) transitioned to running. Waiting for 1 more server.

            All servers are now running.
            wls:/myDomain/serverConfig>
```

shutdown

Command Category: Life Cycle Commands

Use with WLST: Online

Description

Gracefully shuts down a running server instance, cluster, or system component. The `shutdown` command waits for all the in-process work to be completed before shutting down the server, cluster, or system component.

To shut down a server to which WLST is connected, use the `shutdown` command without any arguments.

When connected to a Managed Server instance, use the `shutdown` command to shut down the only Managed Server instance to which WLST is connected. You cannot shut down another server while connected to a Managed Server instance.

> **Note:**
>
> To shut down a system component (for example, Oracle HTTP Server or Oracle Data Integrator system component), WLST must be connected to the Administration Server.

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

Syntax

```
shutdown([name], [entityType], [ignoreSessions], [timeOut], [force], [block],
[properties], [waitForAllSessions])
```

Argument	Definition
name	Optional. Name of the server, cluster, or system component to shut down. This argument defaults to the server to which WLST is connected.
entityType	Optional. Type `Server`, `Cluster`, or `SystemComponent`. If not specified, WLST looks for a server, cluster, or system component with the specified name.
ignoreSessions	Optional. Boolean value specifying whether WLST must drop all HTTP sessions immediately, or wait for HTTP sessions to complete or time out while shutting down. This argument defaults to `false`, indicating that non-persisted HTTP sessions must complete or time out.
timeOut	Optional. Time (in seconds) that WLST waits for subsystems to complete in-process work and suspend themselves before shutting down the server. This argument defaults to 0 seconds, indicating that there is no timeout.
force	Optional. Boolean value specifying whether WLST must terminate a server instance or a cluster without waiting for the active sessions to complete. This argument defaults to `false`, indicating that all active sessions must complete before shut down.
block	Optional. Boolean value specifying whether WLST must block user interaction until the server is shut down. This argument defaults to `true`, indicating that user interaction is blocked until the operation completes. If set to `false`, WLST returns control to the user after issuing the command and assigns the task MBean associated with the current task to a variable that you can use to check its status. If you are importing WLST as a Jython module, as described in Importing WLST as a Jython Module in *Understanding the WebLogic Scripting Tool*, *`block`* is always set to `true`.

Argument	Definition
properties	Optional. Applies only if `entityType` is `SystemComponent`. Properties value specifying properties to pass to the system component.
waitForAllSessions	Optional. Boolean value specifying whether WLST must wait for all HTTP sessions to complete while shutting down. This argument defaults to `false`, which causes WLST to wait for only non-persisted HTTP sessions to complete.

Example

The following example instructs WLST to shut down the server to which you are connected:

```
wls:/mydomain/serverConfig> shutdown()
Shutting down the admin server that you are currently connected to .......
Disconnected from weblogic server: AdminServer
```

The following example instructs WLST to wait 1000 seconds for HTTP sessions to complete or time out (at 1000 seconds) before shutting down `myserver`:

```
wls:/mydomain/serverConfig> shutdown('myserver','Server','false',1000,
block='false')
```

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

```
wls:/mydomain/serverConfig> shutdown('MServer1','Server','true',1200)
Shutting down a managed server that you are connected to ...
Disconnected from weblogic server: MServer1
```

The following example instructs WLST to shut down the cluster `mycluster`:

```
wls:/mydomain/serverConfig> shutdown('mycluster','Cluster')
Shutting down the cluster with name mycluster
Shutdown of cluster mycluster has been issued, please
refer to the logs to check if the cluster shutdown is successful.
Use the state(<server-name>) or state(<cluster-name>,"Cluster")
to check the status of the server or cluster
wls:/mydomain/serverConfig> state('mycluster','Cluster')
There are 3 server(s) in cluster: mycluster

States of the servers are
MServer1---SHUTDOWN
MServer2---SHUTDOWN
MServer3---SHUTDOWN
wls:/mydomain/serverConfig>
```

softRestart

Command Category: Life Cycle Commands

Use with WLST: Online

Description

Restarts a running system component. WLST must be connected to the Administration Server to restart a system component.

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

Syntax

```
softRestart(name, [block], [properties])
```

Argument	Definition
`name`	Name of the system component to restart.
`block`	Optional. Boolean value specifying whether WLST should block user interaction until the server is restarted. This argument defaults to `true`, indicating that user interaction is blocked until the operation completes. If set to `false`, WLST returns control to the user after issuing the command and assigns the task MBean associate with the current task to a variable that you can use to check its status. If you are importing WLST as a Jython module, as described in Importing WLST as a Jython Module in *Understanding the WebLogic Scripting Tool*, *`block`* is always set to `true`.
`properties`	Optional. Properties value specifying properties to pass to the system component.

Example

The following example restarts a system component call ohs1.

```
wls:/mydomain/serverConfig> softRestart('ohs1', block='true')
Restarting the system component with name ohs1 ...
System component with name ohs1 restarted successfully
wls:/mydomain/serverConfig>
```

start

Command Category: Life Cycle Commands

Use with WLST: Online

Description

Starts a Managed Server instance, cluster, or system component using Node Manager. WLST must be connected to the Administration Server. If only starting Managed Servers then WLST can just be connected to the Node Manager.

See Node Manager Commands.

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

Syntax

```
start(name, [type], [listenAddress], [port], [block], [properties])
```

Argument	Definition
`name`	Name of the Managed Server, cluster, or system component to start.
`type`	Optional. Type is `Server`, `Cluster`, or `SystemComponent`. If not specified, WLST will look for a server, cluster, or system component with the specified name.
`listenAddress`	Optional. Listen address of the server instance. If not specified, this defaults to `localhost`. This is ignored if type Cluster or SystemComponent is specified.

Argument	Definition
port	Optional. Listen port of the server instance. If not specified, this defaults to `7001`. This is ignored if type Cluster or SystemComponent is specified.
block	Optional. Boolean value specifying whether WLST should block user interaction until the server or cluster is started. This argument defaults to `true`, indicating that user interaction is blocked until the operation completes. If set to `false`, WLST returns control to the user after issuing the command and assigns the task MBean associated with the current task to a variable that you can use to check its status. If you are importing WLST as a Jython module, as described Importing WLST as a Jython Module in *Understanding the WebLogic Scripting Tool*, *`block`* is always set to `true`.
properties	Optional. Applies only if `type` is `SystemComponent`. Properties value specifying properties to pass to the system component.

Example

The following example instructs Node Manager to start a Managed Server instance; the listen address is `localhost` and listen port is `8801`. WLST returns control to the user after issuing this command, as `block` is set to `false`.

```
wls:/mydomain/serverConfig> start('myserver', 'Server', block='false')
Starting server myserver ...
The server start status task for server myserver is assigned to variable
myserverTask
You can call the getStatus(), getError(), getDescription() or isRunning()
methods on this variable to determine the status of your server start
wls:/mydomain/serverConfig>
```

The following example instructs Node Manager to start a cluster. WLST block user interaction until the cluster is started, as `block` defaults to `true`.

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

startServer

Command Category: Life Cycle Commands

Use with WLST: Online or Offline

Description

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

Note:

You cannot use `startServer` to start an integrated WebLogic Administration Server (that is, an Administration Server for a Fusion Middleware Suite product installed in an *`ORACLE_HOME`* directory or if you chose the templates for Fusion Middleware (JRF and Enterprise Manager) during Domain creation).

To start the Administration server for a Fusion Middleware Suite product other than WebLogic Server, use either of the following methods:

- Execute the server startup script for the associated WebLogic domain.
- Start the server using Node Manager. If you use this method, make sure that the `startScriptEnabled` property is set to `true` in Node Manager.

Syntax

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

Argument	Definition
`adminServerName`	Optional. Name of the Administration Server to start. This argument defaults to `myserver`.
`domainName`	Optional. Name of the WebLogic domain to which the Administration Server belongs. This argument defaults to `mydomain`.
`url`	Optional. URL of the Administration Server. The URL supplied with the startServer command will override the listen address and port specified in the `config.xml` file. If not specified on the command line or in the `config.xml` file, this argument defaults to `t3://localhost:7001`.
`username`	Optional. Username used to connect WLST to the server.
`password`	Optional. Password used to connect WLST to the server.
`domainDir`	Optional. Domain directory in which the Administration Server is being started. This argument defaults to the directory from which you started WLST.
`block`	Optional. Boolean value specifying whether WLST blocks user interaction until the server is started. When *`block`* is set to `false`, WLST returns control to the user after issuing the command. This argument defaults to `true`, indicating that user interaction is blocked. If you are importing WLST as a Jython module, as described in Importing WLST as a Jython Module in *Understanding the WebLogic Scripting Tool*, *`block`* is always set to `true`.
`timeout`	Optional. Time (in milliseconds) that WLST waits for the server to start before canceling the operation. The default value is 60000 milliseconds. This argument is only applicable when *`block`* is set to `true`.
`serverLog`	Optional. Location of the server log file. This argument defaults to `stdout`.
`systemProperties`	Optional. System properties to pass to the server process. System properties should be specified as comma-separated name-value pairs, and the name-value pairs should be separated by equals sign (=).

Argument	Definition
jvmArgs	Optional. JVM arguments to pass to the server process. Multiple arguments can be specified, separated by commas.
spaceAsJvmArgsDelimiter	Optional. Boolean value specifying whether JVM arguments are space delimited. The default value is `false`.

Example

The following example starts the Administration Server named `demoServer` in the `demoDomain`.

```
wls:/offline> startServer('demoServer','demoDomain','t3://localhost:8001',
'myweblogic','wlstdomain','c:/mydomains/wlst','false', 60000,
jvmArgs='-XX:MaxPermSize=75m, -Xmx512m, -XX:+UseParallelGC')
wls:/offline>
```

suspend

Command Category: Life Cycle Commands

Use with WLST: Online

Description

Suspends a running server. This command moves a server from the `RUNNING` state to the `ADMIN` state. See Understanding Server Life Cycle in *Administering Server Startup and Shutdown for Oracle WebLogic Server*.

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

Syntax

```
suspend([sname], [ignoreSessions], [timeOut], [force], [block])
```

Argument	Definition
sname	Optional. Name of the server to suspend. The argument defaults to the server to which WLST is currently connected.
ignoreSessions	Optional. Boolean value specifying whether WLST should drop all HTTP sessions immediately or wait for HTTP sessions to complete or time out while suspending. This argument defaults to `false`, indicating that HTTP sessions must complete or time out.
timeOut	Optional. Time (in seconds) the WLST waits for the server to complete in-process work before suspending the server. This argument defaults to 0 seconds, indicating that there is no timeout.
force	Optional. Boolean value specifying whether WLST should suspend the server without waiting for active sessions to complete. This argument defaults to `false`, indicating that all active sessions must complete before suspending the server.

Argument	Definition
`block`	Optional. Boolean value specifying whether WLST blocks user interaction until the server is started. This argument defaults to `false`, indicating that user interaction is not blocked. In this case, WLST returns control to the user after issuing the command and assigns the task MBean associated with the current task to a variable that you can use to check its status. If you are importing WLST as a Jython module, as described in Importing WLST as a Jython Module in *Understanding the WebLogic Scripting Tool*, *`block`* is always set to `true`.

Example

The following example suspends a Managed Server instance:

```
wls:/mydomain/serverConfig> suspend('managed1')
Server 'managed1' suspended successfully.
wls:/mydomain/serverConfig>
```

Node Manager Commands

Use the WLST Node Manager commands to start, shut down, restart, and monitor WebLogic Server instances. Table 2-11 lists and summarizes these commands.

Note:

Unless otherwise indicated, Node Manager must be running before you can execute the commands within this category.

See Using Node Manager in the *Administering Node Manager for Oracle WebLogic Server*.

Table 2-11 Node Manager Commands for WLST Configuration

This command...	Enables you to...	Use with WLST...
getNodeManagerHome	Get the Node Manager home.	Offline
getNodeManagerType	Get the Node Manager type.	Offline
getNodeManagerUpgradeOverwriteDefault	Get the value of the Node Manager upgrade overwrite default flag.	Offline
getNodeManagerUpgradeType	Get the Node Manager upgrade type used for Node Manager upgrade.	Offline
getOldNodeManagerHome	Get the old Node Manager home used for Node Manager upgrade.	Offline
nm	Determine whether WLST is connected to Node Manager.	Online
nmConnect	Connect WLST to Node Manager to establish a session.	Online or Offline
nmDisconnect	Disconnect WLST from a Node Manager session.	Online or Offline

Table 2-11 (Cont.) Node Manager Commands for WLST Configuration

This command...	Enables you to...	Use with WLST...
nmEnroll	Enable the Node Manager on the current computer to manage servers in a specified WebLogic domain.	Online
nmExecScript	Execute the named script using the connected Node Manager.	Online
nmGenBootStartupProps	Generate the Node Manager property files, `boot.properties` and `startup.properties`, for the specified server.	Online
nmKill	Kill the specified server instance that was started with Node Manager.	Online or Offline
nmLog	Return the Node Manager log.	Online or Offline
nmRestart	Restart the Node Manager instance.	Online
nmServerLog	Return the server output log of the server that was started with Node Manager.	Online or Offline
nmServerStatus	Return the status of the server that was started with Node Manager.	Online or Offline
nmSoftRestart	Restart the specified System Component server instance.	Online or Offline
nmStart	Start a server in the current WebLogic domain using Node Manager.	Online or Offline
nmVersion	Return the Node Manager server version.	Online or Offline
startNodeManager	Start Node Manager on the same computer that is running WLST.	Online or Offline
stopNodeManager	Stop Node Manager.	Online or Offline

getNodeManagerHome

Command Category: Control Commands

Use with WLST: Offline

Description

Gets the Node Manager home directory for a domain. In the event of an unsupported operation, the command returns a `WLSTException`. Node Manager does not have to be running in order to execute this command.

Syntax

```
getNodeManagerHome()
```

Example

The following example returns the Node Manager home directory:

```
wls:/offline/base_domain>getNodeManagerHome()
'C:\\domains\\my_domain\\nodemanager'
```

getNodeManagerType

Command Category: Control Commands

Use with WLST: Offline

Description

Gets the Node Manager type for a domain (`PerDomainNodeManager`, `CustomLocationNodeManager`, or `ManualNodeManagerSetup`). In the event of an unsupported operation, the command returns a `WLSTException`. Node Manager does not have to be running in order to execute this command.

For information on Node Manager types, see Default Node Manager Configuration in *Administering Node Manager for Oracle WebLogic Server*.

Syntax

```
getNodeManagerType()
```

Example

The following example returns the Node Manager type for the domain. In this case, it is a `PerDomainNodeManager`.

```
wls:/offline/base_domain>getNodeManagerType()
'PerDomainNodeManager'
```

getNodeManagerUpgradeOverwriteDefault

Command Category: Control Commands

Use with WLST: Offline

Description

Gets the value of the Node Manager upgrade overwrite default flag. In the event of an unsupported operation, the command returns a `WLSTException`. Node Manager does not have to be running in order to execute this command.

Syntax

```
getNodeManagerUpgradeOverwriteDefault()
```

Example

The following command returns a value of true or false.

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

getNodeManagerUpgradeType

Command Category: Control Commands

Use with WLST: Offline

Description

Gets the Node Manager upgrade type to be used for Node Manager upgrade during domain reconfiguration. In the event of an unsupported operation, the command returns a `WLSTException`. Node Manager does not have to be running in order to execute this command.

Syntax

```
getNodeManagerUpgradeType()
```

Example

The following example returns a value of `New` or `Migrate`.

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

getOldNodeManagerHome

Command Category: Control Commands

Use with WLST: Offline

Description

Gets the old Node Manager home to be used for Node Manager upgrade during domain reconfiguration. In the event of an unsupported operation, the command returns a `WLSTException`. Node Manager does not have to be running in order to execute this command.

Syntax

```
getOldNodeManagerHome
```

Example

In this example, the command returns the Node Manager home directory (/scratch/domain/nodemanager) that was used when the domain was reconfigured during an upgrade.

```
wls:/offline/base_domain>getOldNodeManagerHome()
/scratch/domains/nodemanager
```

nm

Command Category: Node Manager Commands

Use with WLST: Online or Offline

Description

Determines whether WLST is connected to Node Manager. Returns `true` or `false` and prints a descriptive message. Node Manager must be running before you can execute this command.

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

Syntax

```
nm()
```

Example

The following example indicates that WLST is currently connected to Node Manager that is monitoring `mydomain`.

```
wls:/mydomain/serverConfig> nm()
Currently connected to Node Manager that is monitoring the domain "mydomain"
wls:/mydomain/serverConfig>
```

The following example indicates that WLST is not currently connected to Node Manager.

```
wls:/mydomain/serverConfig> nm()
Not connected to any Node Manager
wls:/mydomain/serverConfig>
```

nmConnect

Command Category: Node Manager Commands

Use with WLST: Online or Offline

Description

Connects WLST to Node Manager to establish a session. After connecting to Node Manager, you can invoke any Node Manager commands via WLST. Node Manager must be running before you can execute this command.

Note:

If you have previously used the `connect` command in the current WLST session, `nmconnect` uses the same user credentials as were used for the `connect` command, unless you specify otherwise.

Once connected, the WLST prompt displays as follows, where *`domainName`* indicates the name of the WebLogic domain that is being managed: `wls:/nm/`*`domainName`*`>`. If you then connect WLST to a WebLogic Server instance, the prompt is changed to reflect the WebLogic Server instance. You can use the `nm` command to determine whether WLST is connected to Node Manager, as described in nm.

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

Syntax

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

nmConnect([userConfigFile, userKeyFile], [host], [port], [domainName], [domainDir],
[nmType], [verbose])
```

Argument	Definition
username	Username of the operator who is connecting WLST to Node Manager. **Note:** When running a server in production mode, you must specify the username and password explicitly on the command line to ensure that the appropriate username and password are used when connecting to Node Manager.
password	Password of the operator who is connecting WLST to Node Manager. **Note:** When running a server in production mode, you must specify the username and password explicitly on the command line to ensure that the appropriate username and password are used when connecting to Node Manager.
host	Optional. Host name of Node Manager. This argument defaults to `localhost`.
port	Optional. Port number of Node Manager. This argument defaults to a value that is based on the Node Manager type, as follows: • For `plain` type, defaults to 5556 • For `rsh` type, defaults to 514 • For `ssh` type, defaults to 22 • For `ssl` type, defaults to 5556
domainName	Optional. Name of the WebLogic domain that you want to manage. This argument defaults to `mydomain`.
domainDir	Optional. Path or directory of the domain on the remote node. This argument defaults to the directory in which WLST was started.
nmType	The Node Manager type. Valid values are: • `plain` for plain socket Java-based implementation **Note:** If you specify `plain` for `nmType`, you must manually set the `SecureListener` parameter in `nodemanager.properties` to `false`. Otherwise, the `nmConnect` command will fail. • `rsh` for RSH implementation • `ssh` for script-based SSH implementation • `ssl` for Java-based SSL implementation This argument defaults to `ssl`.
verbose	Optional. Boolean value specifying whether WLST connects to Node Manager in verbose mode. This argument defaults to `false`, disabling verbose mode.
userConfigFile	Optional. Name and location of a user configuration file which contains an encrypted username and password. Use the following syntax: `userConfigFile='file-system-path'`. When you create a user configuration file, the `storeUserConfig` command uses a key file to encrypt the username and password. Only the key file that encrypts a user configuration file can decrypt the username and password. (See storeUserConfig.)
userKeyFile	Optional. Name and location of the key file that is associated with the specified user configuration file and is used to decrypt it. Use the following syntax: `userKeyFile='file-system-path'`. (See storeUserConfig.)

Example

The following example connects WLST to Node Manager to monitor the `oamdomain` domain using the default host and port numbers and `plain` Node Manager type.

```
wls:/myserver/serverConfig> nmConnect('username', 'password', 'localhost',
'5555', 'oamdomain', 'c:/Oracle/Middleware/user_projects/domains/oamdomain','ssl')
Connecting to Node Manager Server ...
Successfully connected to Node Manager.
wls:/nm/oamdomain>
```

The following example connects WLST to a Node Manager Server instance using a user configuration and key file to provide user credentials.

```
wls:/myserver/serverConfig> nmConnect(userConfigFile='
c:/myfiles/myuserconfigfile.secure',
userKeyFile='c:/myfiles/myuserkeyfile.secure',
host='172.18.137.82', port=26106, domainName='mydomain',
domainDir='c:/myfiles/mydomain', nmType='ssl')
Connecting to Node Manager Server ...
Successfully connected to Node Manager.
wls:/nm/mydomain>
```

nmDisconnect

Command Category: Node Manager Commands

Use with WLST: Online or Offline

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

Description

Disconnects WLST from a Node Manager session.

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

Syntax

```
nmDisconnect()
```

Example

The following example disconnects WLST from a Node Manager session.

```
wls:/nm/oamdomain> nmDisconnect()
Successfully disconnected from Node Manager
wls:/myserver/serverConfig>
```

nmEnroll

Command Category: Node Manager Commands

Use with WLST: Online

Description

Enrolls the machine on which WLST is currently running. WLST must be connected to an Administration Server to run this command; WLST does not need to be connected to Node Manager.

From the Administration Server, this command downloads the Node Manager secret file (`nm_password.properties`), which contains the encrypted user name and password that is used for server authentication.

> **Note:**
>
> In previous releases, this command would also download the `SerializedSystemIni.dat` file from the Administration Server. However, now, if it does not already exist, you must manually copy the file from the Administration Server domain directory or an existing server in the domain *before* running the `nmEnroll` command. The file location should be `${DOMAIN_HOME}/security/SerializedSystemIni.dat`.

This command also updates the `nodemanager.domains` file, under the *`NodeManagerHome`* directory, with the domain information. For the Java-based Node Manager, this file is typically located under *`domain_home`*`\nodemanager`. For the script-based Node Manager, this file's default location is *`WL_HOME`*`/common/nodemanager`, where *`WL_HOME`* is the location in which you installed WebLogic Server, for example, *`ORACLE_HOME`*`/wlserver`.

You must run this command once per WebLogic domain per machine unless that domain shares the root directory of the Administration Server.

If the machine is already enrolled when you run this command, the Node Manager secret file (`nm_password.properties`) is refreshed with the latest information from the Administration Server.

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

Syntax

```
nmEnroll([domainDir], [nmHome])
```

Argument	Definition
`domainDir`	Optional. Path or directory of the domain on the remote node. This argument defaults to the directory in which WLST was started.
`nmHome`	Optional. Path to the Node Manager home. The `nodemanager.domains` file, containing the domain information, is written to this directory. This argument defaults to *`WL_HOME`*`/common/nodemanager`, where *`WL_HOME`* refers to the top-level installation directory for WebLogic Server.

Example

The following example enrolls the current machine with Node Manager and saves the Node Manager secret file (`nm_password properties`) to `c:/Oracle/Middleware/`

`mydomain/common/nodemanager/nm_password.properties`. The `nodemanager.domains` file is written to *`WL_HOME`*`/common/nodemanager` by default.

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

nmExecScript

Command Category: Node Manager Commands

Use with WLST: Online

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

Description

Executes the named script using the connected Node Manager.

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

Syntax

```
nmExecScript(scriptName, [scriptDir], [scriptProps], [writer], [timeout])
```

Argument	Definition
`scriptName`	Name of the script to be executed. This must be the name of the script without any path information and must match with the name of the file on the disk.
`scriptDir`	Optional. Directory under the domain directory where the Node Manager can find the script. This must be one of the well known script locations. By default, the NodeManager checks the `mydomain/bin/service_migration` directory (where `mydomain` is a domain-specific directory).
`scriptProps`	Optional. Properties supplied to the script as environment variables. This argument defaults to `None`.
`writer`	Optional. The `java.io.Writer` object to which the script output is written. This argument defaults to the WLST writer.
`timeout`	Optional. The number of milliseconds to wait for the script to finish execution. After waiting for the specified number of milliseconds, the Node Manager attempts to cancel the script process, collects any available output, and cleans up. The Node Manager returns the timeout as an error. By default, the Node Manager waits for the script to complete its execution.

Example

The following example executes the script named `foo.sh` using the connected Node Manager, and waits 20000 milliseconds for the script to finish execution.

```
wls:/nm/oamdomain> nmExecScript('foo.sh', timeout=20000)
output from script
wls:/myserver/serverConfig>
```

nmGenBootStartupProps

Command Category: Node Manager Commands

Use with WLST: Online

Description

Generates the Node Manager properties files, `boot.properties` and `startup.properties`, for the specified server. The Node Manager properties files are stored relative to the root directory of the specified server. The target root directory must be on the same machine on which you are running the command.

You must specify the name of a server; otherwise, the command will fail.

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

Syntax

```
nmGenBootStartupProps(serverName)
```

Argument	Definition
`serverName`	Name of the server for which Node Manager property files are generated.

Example

The following example generates `boot.properties` and `startup.properties` in the root directory of the specified server, `ms1`.

```
wls:/mydomain/serverConfig> nmGenBootStartupProps('ms1')
Successfully generated boot.properties at
c:\Oracle\Middleware\mydomain\servers\ms1\data\nodemanager\boot.properties
Successfully generated startup.properties at
c:\Oracle\Middleware\mydomain\servers\ms1\data\nodemanager\startup.properties
wls:/mydomain/serverConfig>
```

nmKill

Command Category: Node Manager Commands

Use with WLST: Online or Offline

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

Description

Kills the specified server instance that was started with Node Manager.

If you do not specify a server name using the *`serverName`* argument, the argument defaults to `myServer`, which must match your server name or the command will fail.

If you attempt to kill a server instance that was not started using Node Manager, the command displays an error.

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

Syntax

```
nmKill([serverName], [serverType], [pluginProps])
```

Argument	Definition
serverName	Optional. Name of the server to be killed. This argument defaults to `myserver`.
serverType	Optional. The type of server to kill. This argument defaults to `WebLogic`. Another valid option is `Coherence`.
pluginProps	Optional. The properties to use to kill the server. Defaults to `None`. Only relevant to plugin-handling system components.

Example

The following example kills the server named `oamserver`.

```
wls:/nm/oamdomain> nmKill('oamserver')
Killing server 'oamserver' ...
Server oamServer killed successfully.
wls:/nm/oamdomain>
```

nmLog

Command Category: Node Manager Commands

Use with WLST: Online or Offline

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

Description

Returns the Node Manager log.

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

Syntax

```
nmLog([writer])
```

Argument	Definition
writer	Optional. `java.io.Writer` object to which you want to stream the log output. This argument defaults to the WLST writer stream.

Example

The following example displays the Node Manager log.

```
wls:/nm/oamdomain> nmLog()
Successfully retrieved the Node Manager log and written.
wls:/nm/oamdomain>
```

nmRestart

Command Category: Node Manager Commands

Use with WLST: Online

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

Description

Restarts the connected Node Manager instance.

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

> **Note:**
>
> If Node Manager is started by any means other than the `startNodeManager` script, the use of the `nmRestart` WLST command to restart Node Manager is not supported. If Node Manager is started from a custom script, or by the `java weblogic.NodeManager` command, using the `nmRestart` command may subsequently fail.

Syntax

```
nmRestart([timeout])
```

Argument	Definition
timeout	Optional. The number of milliseconds that WLST waits for the client to reconnect to the NodeManager after restart. Once the number of specified milliseconds has been exceeded, the command returns an error indicating timeout. By default, the command blocks until the client has successfully connected to the restarted Node Manager.

Example

The following example specifies the timeout and restarts the Node Manager instance.

```
wls:/nm/oamdomain> nmRestart(5000)
Restarted Node Manager Process successfully
wls:/myserver/serverConfig>
```

nmServerStatus

Command Category: Node Manager Commands

Use with WLST: Online or Offline

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

Description

Returns the status of the server that was started with Node Manager.

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

Syntax

```
nmServerStatus([serverName], [serverType])
```

Argument	Definition
`serverName`	Optional. Name of the server for which you want to display the status. This argument defaults to `myserver`.
`serverType`	Optional. The type of server to start. This argument defaults to `WebLogic`. Another valid option is `Coherence`.

Example

The following example displays the status of the server named `oamserver`, which was started with Node Manager.

```
wls:/nm/oamdomain> nmServerStatus('oamserver')
RUNNING
wls:/nm/oamdomain>
```

nmServerLog

Command Category: Node Manager Commands

Use with WLST: Online or Offline

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

Description

Returns the server output log of the server that was started with Node Manager.

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

Syntax

```
nmServerLog([serverName], [writer], [serverType])
```

Argument	Definition
`serverName`	Optional. Name of the server for which you want to display the server output log. This argument defaults to `myserver`.
`writer`	Optional. `java.io.Writer` object to which you want to stream the log output. This argument defaults to the WLSTInterpreter standard out, if not specified.
`serverType`	Optional. The type of server to start. This argument defaults to `WebLogic`. Another valid option is `Coherence`.

Example

The following example displays the server output log for the `oamserver` server and writes the log output to `myWriter`.

```
wls:/nm/oamdomain> nmServerLog('oamserver',myWriter)
Successfully retrieved the server log and written.
wls:/nm/oamdomain>
```

nmSoftRestart

Command Category: Node Manager Commands

Use with WLST: Online or Offline

Description

Restarts the specified system component server instance that was started with Node Manager. This command can be used only with Oracle system components (such as OHS) that are currently supported by Node Manager and WLST.

If you do not specify a server name using the `serverName` argument, the argument defaults to `myServer`, which must match your server name or the command will fail.

If you attempt to restart a system component server instance that was not started using Node Manager, the command displays an error. WLST must be connected to Node Manager to run this command.

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

Syntax

```
nmSoftRestart([serverName], serverType, [pluginProps])
```

Argument	Definition
serverName	Optional. Name of the system component server to be restarted. Defaults to `myServer`.
serverType	Required. The type of server to start (for example, OHS). Refer to the Administration Guide or other documentation for the system component to determine the appropriate value to use for this argument.
pluginProps	Optional. The properties to use to kill the server. Defaults to `None`. Only relevant to plugin-handling system components.

Example

The following example starts a System Component server instance named `ohsServer`.

```
wls:/nm/oamdomain> nmSoftRestart('ohsServer', 'OHS')
Restarting server 'ohsServer' ...
Server ohsServer restarted successfully.
wls:/nm/oamdomain>
```

nmStart

Command Category: Node Manager Commands

Use with WLST: Online or Offline

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

Description

Starts a server in the current WebLogic domain using Node Manager.

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

Note:

boot.properties must exist in order to start a server with nmStart. If this is the first time you are starting a server or the first time you are using Node Manager, you must manually create it or run the nmGenBootStartupProps command to generate boot.properties and startup.properties files for the server.

Alternatively, you can use the nmStartprops argument to provide user credentials (after connecting to Node Manager):

```
prps = makePropertiesObject("AdminURL=http://
listen_address:listen_port;Username=username;Password=password")
nmStart("managed1",props=prps)
```

Syntax

```
nmStart([serverName], [domainDir], [props], [writer], [serverType], [pluginProps])
```

Argument	Definition
serverName	Optional. Name of the server to be started.
domainDir	Optional. Domain directory of the server to be started. This argument defaults to the directory from which you started WLST.
props	Optional. System properties to apply to the new server.
writer	Optional. java.io.Writer object to which the server output is written. This argument defaults to the WLST writer.
serverType	Optional. The type of server to start. This argument defaults to WebLogic. Another valid option is Coherence.
pluginProps	Optional. The properties to use to start the server. Defaults to None. Only relevant to plugin-handling system components.

Example

The following example starts the managed1 server in the current WebLogic domain using Node Manager. In this example, the prps variable stores the system property settings and is passed to the command using the props argument.

```
wls:/nm/mydomain> prps = makePropertiesObject("AdminURL=http://
listen_address:listen_port;Username=username;Password=password
;weblogic.ListenPort=8001")
wls:/nm/mydomain> nmStart("managed1",props=prps)
Starting server managed1 ...
Server managed1 started successfully
wls:/nm/mydomain>
```

The following example starts the Administration Server in the specified WebLogic domain using Node Manager.

```
wls:/nm/mydomain> nmStart("AdminServer")
Starting server AdminServer...
Server AdminServer started successfully
wls:/nm/mydomain>
```

nmVersion

Command Category: Node Manager Commands

Use with WLST: Online or Offline

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

Description

Returns the Node Manager version.

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

Syntax

```
nmVersion()
```

Example

The following example displays the Node Manager version.

```
wls:/nm/oamdomain> nmVersion()
The Node Manager version that you are currently connected to is 9.0.0.0
wls:/nm/oamdomain>
```

startNodeManager

Command Category: Node Manager Commands

Use with WLST: Online or Offline

Description

Starts Node Manager on the same computer that is running WLST.

Note:

In production environments, Oracle recommends that you do *not* use the `startNodeManager` command to start Node Manager. The recommended approach is to install Node Manager as a service or daemon, or to use the startNodeManager script (`startNodeManager.sh` or `startNodeManger.cmd`).

If Node Manager is already running when you invoke the `startNodeManager` command, the following message is displayed:

```
A Node Manager has already been started.
Cannot start another Node Manager process via WLST
```

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

Syntax

```
startNodeManager([verbose], [block], [timeout], [jvmArgs], [nmProperties],
[nmConnectionOptions])
```

Argument	Definition
verbose	Optional. Boolean value specifying whether WLST starts Node Manager in verbose mode. This argument defaults to `false`, disabling verbose mode.
block	Optional. A boolean value that specifies whether WLST should block until it successfully connects to the NodeManager or fails to connect to NodeManager within the time specified by the `timeout` argument. This option defaults to `false`. When this option is set to `true`, you must provide the username, password, host, port, domain name, and Node Manager type to use for nmConnect.
timeout	Optional. Number of milliseconds to wait for WLST to connect to the NodeManager. This option defaults to 120000 milliseconds.
jvmArgs	Optional. JVM arguments to pass to the server process. Multiple arguments can be specified, separated by commas. See below for a usage example for this argument.
nmProperties	Optional. Comma-separated list of Node Manager properties, specified as name-value pairs. Node Manager properties include, but are not limited to, the following: `NodeManagerHome`, `ListenAddress`, `ListenPort`, and `PropertiesFile`.
nmConnectionOptions	Optional when block is false and required when block is true. A comma-separated list ofnmConnect options. The following arguments are required: `username`, `password`, `host`, `port`, and `domainName` The following arguments are optional: `domainDir` and `nmType`

Example

The following example starts Node Manager using C:/Oracle/Middleware/wlserver/common/nodemanager as the Node Manager home and port `6666` as the Node Manager listen address on `myhost`. The JVM arguments set the JVM starting and maximum memory for Node Manager.

```
wls:/mydomain/serverConfig> startNodeManager(block='true',
timeout=30000,verbose='true', NodeManagerHome='c:/Oracle/Middleware/wlserver/common/
nodemanager', ListenPort='6666', ListenAddress='myhost', jvmArgs='-Xms24m,-Xmx64m')
Launching Node Manager ...
Successfully launched the Node Manager.
The Node Manager process is running independent of the WLST process
Exiting WLST will not stop the Node Manager process. Please refer
to the Node Manager logs for more information.
The Node Manager logs will be under c:\Oracle\Middleware\wlserver\common\nodemanager.
wls:/mydomain/serverConfig>
```

stopNodeManager

Command Category: Node Manager Commands

Use with WLST: Online or Offline

Description

Stops the Node Manager process.

> **Note:**
>
> In order to stop the Node Manager process, you must have either started Node Manager with `startNodeManager`, or Node Manager must have been started with the property `QuitEnabled=true`. You can configure this property in `$WLS_HOME/common/nodemanager.properties`. This allows you to connect to the Node Manager to shut it down.

If the Node Manager is not running when you invoke the `stopNodeManager` command, the following message is displayed:

```
Cannot stop the Node Manager unless you are connected to it.
```

Syntax

```
stopNodeManager()
```

Example

The following example stops the Node Manager process for the `base_domain` domain.

```
wls:/nm/base_domain> stopNodeManager()
Stopped Node Manager Process successfully
wls:/offline>
```

Tree Commands

Use the WLST tree commands to navigate among MBean hierarchies. Table 2-12 lists and summarizes these commands.

Table 2-12 Tree Commands for WLST Configuration

Use this command...	To...	Use with WLST...
custom	Navigate to the root of custom MBeans that are registered in the server.	Online
domainConfig	Navigate to the last MBean to which you navigated in the domain configuration hierarchy or to the root of the hierarchy, `DomainMBean`.	Online
domainCustom	Navigate to the root of custom MBeans that are registered in the Domain Runtime MBean Server	Online
domainRuntime	Navigate to the last MBean to which you navigated in the domain runtime hierarchy or to the root of the hierarchy, `DomainRuntimeMBean`.	Online
edit	Navigate to the last MBean to which you navigated in the edit configuration MBean hierarchy or to the root of the hierarchy, `DomainMBean`.	Online
editCustom	Navigate to the root of custom MBeans.	Online
jndi	Navigates to the JNDI tree for the server to which WLST is currently connected.	Online

Table 2-12 (Cont.) Tree Commands for WLST Configuration

Use this command...	To...	Use with WLST...
serverConfig	Navigate to the last MBean to which you navigated in the configuration MBean hierarchy or to the root of the hierarchy, `DomainMBean`.	Online
serverRuntime	Navigate to the last MBean to which you navigated in the runtime MBean hierarchy or to the root of the hierarchy, `ServerRuntimeMBean`.	Online

custom

Command Category: Tree Commands

Use with WLST: Online

Description

Navigates to the root of custom MBeans that are registered in the Runtime MBean Server. WLST navigates, interrogates, and edits custom MBeans as it does domain MBeans; however, custom MBeans cannot use the `cmo` variable because a stub is not available.

Note:

When navigating to the `custom` tree, WLST queries all MBeans in the compatibility MBean server, the runtime MBean server, and potentially the JVM platform MBean server to locate the custom MBeans. Depending on the number of MBeans in the current WebLogic domain, this process may take a few minutes, and WLST may not return a prompt right away. Oracle recommends that you specify a JMX query Object Name Pattern to limit the amount of searching performed.

The `custom` command is available when WLST is connected to an Administration Server instance or a Managed Server instance. When connected to a WebLogic Integration or WebLogic Portal server, WLST can interact with all of the WebLogic Integration or WebLogic Portal server MBeans.

See Instrumenting and Registering Custom MBeans in *Developing Manageable Applications Using JMX for Oracle WebLogic Server*.

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

Note:

You can also navigate to custom MBeans on the Domain Runtime MBean Server using the `domainCustom()` command. See domainCustom.

Syntax

```
custom([objectNamePattern])
```

Argument	Definition
ObjectNamePattern	A JMX query pattern, such as `sip:*`. The default value is null or `*:*`.

Example

The following example navigates from the configuration MBean hierarchy to the custom MBean hierarchy on an Administration Server instance.

```
wls:/mydomain/serverConfig> custom()
Location changed to custom tree. This is a writeable tree with No root. For more
help, use help('custom')
wls:/mydomain/custom>
```

domainConfig

Command Category: Tree Commands

Use with WLST: Online

Description

Navigates to the last MBean to which you navigated in the domain Configuration hierarchy or to the root of the hierarchy, `DomainMBean`. This read-only hierarchy stores the configuration MBeans that represent your current WebLogic domain.

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

Syntax

```
domainConfig()
```

Example

The following example navigates from the configuration MBean hierarchy to the WebLogic domain Configuration hierarchy on an Administration Server instance.

```
wls:/mydomain/serverConfig> domainConfig()
Location changed to domainConfig tree. This is a read-only tree with DomainMBean
as the root.
For more help, use help('domainConfig')
wls:/mydomain/domainConfig> ls()
dr--   AppDeployments
dr--   BridgeDestinations
dr--   Clusters
dr--   CustomResources
dr--   DeploymentConfiguration
dr--   Deployments
dr--   EmbeddedLDAP
dr--   ErrorHandlings
dr--   FileStores
dr--   InternalAppDeployments
dr--   InternalLibraries
dr--   JDBCDataSourceFactories
dr--   JDBCStores
```

```
dr--   JDBCSystemResources
dr--   JMSBridgeDestinations
dr--   JMSInteropModules
dr--   JMSServers
dr--   JMSSystemResources
...
wls:/mydomain/domainConfig>
```

domainCustom

Command Category: Tree Commands

Use with WLST: Online

Description

Navigates to the domain custom tree of custom MBeans that are registered in the Domain Runtime MBean Server. WLST navigates, interrogates, and edits domain custom MBeans as it does domain MBeans; however, domain custom MBeans cannot use the `cmo` variable because a stub is not available.

> **Note:**
>
> When navigating to the domainCustom tree, WLST queries all MBeans in the Domain Runtime MBean Server, the Runtime MBean Servers on each server, and potentially the JVM platform MBean server to locate the custom MBeans. Depending on the number of MBeans in the current WebLogic domain, this process may take a few minutes, and WLST may not return a prompt right away. Oracle recommends that you specify a JMX query Object Name Pattern to limit the amount of searching performed.

The `domainCustom` command is available only when WLST is connected to an Administration Server instance.

See Understanding WebLogic Server MBeans in *Developing Custom Management Utilities Using JMX for Oracle WebLogic Server*.

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

Syntax

```
domainCustom(ObjectNamePattern)
```

Argument	Definition
`ObjectNamePattern`	A JMX query pattern, such as `sip:*`. The default value is null or `*:*`.

Example

The following example navigates from the configuration MBean hierarchy to the domain custom MBean hierarchy on an Administration Server instance:

```
wls:/mydomain/serverConfig> domainCustom()
Location changed to domain custom tree. This is a writeable tree with No root. For
more help, use help('domainCustom').
```

```
wls:/mydomain/domainCustom
```

domainRuntime

Command Category: Tree Commands

Use with WLST: Online

Description

Navigates to the last MBean to which you navigated in the domain Runtime hierarchy or to the root of the hierarchy, `DomainRuntimeMBean`. This read-only hierarchy stores the runtime MBeans that represent your current WebLogic domain.

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

Syntax

```
domainRuntime()
```

Example

The following example navigates from the configuration MBean hierarchy to the domain Runtime hierarchy on an Administration Server instance.

```
wls:/mydomain/serverConfig> domainRuntime()
wls:/mydomain/domainRuntime> ls()
dr--   AppRuntimeStateRuntime
dr--   DeployerRuntime
dr--   DomainServices
dr--   LogRuntime
dr--   MessageDrivenControlEJBRuntime
dr--   MigratableServiceCoordinatorRuntime
dr--   MigrationDataRuntimes
dr--   SNMPAgentRuntime
dr--   ServerLifeCycleRuntimes
dr--   ServerRuntimes
dr--   ServerServices

-r--   ActivationTime                             Mon Aug 01 11:41:25 EDT 2005
-r--   Clusters                                   null
-r--   MigrationDataRuntimes                      null
-r--   Name                                       sampleMedRecDomain
-rw-   Parent                                     null
-r--   SNMPAgentRuntime                           null
-r--   Type                                       DomainRuntime
-r-x   restartSystemResource                      Void :
       WebLogicMBean(weblogic.management.configuration.SystemResourceMBean)
wls:/mydomain/domainRuntime>
```

edit

Command Category: Tree Commands

Use with WLST: Online

Description

Navigates to the last MBean to which you navigated in the edit configuration MBean hierarchy or to the root of the hierarchy, `DomainMBean`. This writable hierarchy stores all of the configuration MBeans that represent your current WebLogic domain.

> **Note:**
>
> To edit configuration beans, you must be connected to an Administration Server. If you connect to a Managed Server, WLST functionality is limited to browsing the configuration bean hierarchy. While you cannot use WLST to change the values of MBeans on Managed Servers, it is possible to use the Management APIs to do so. Oracle recommends that you change only the values of configuration MBeans on the Administration Server. Changing the values of MBeans on Managed Servers can lead to an inconsistent domain configuration.
>
> See Using WLST Online to Update an Existing Domain in *Understanding the WebLogic Scripting Tool*.

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

Syntax

```
edit([editSessionName])
```

Argument	Definition
`editSessionName`	Optional. The name of the edit session to the context of which you want to navigate. In case an edit session with the specified name does not exist, it gets automatically created. If this argument is not specified, the default edit session is used.

Example

The following example illustrates how to navigate from the server configuration MBean hierarchy to the editable copy of the domain configuration MBean hierarchy, in an Administration Server instance. It then calls out the name of a specific edit session, `mySample`. Since this edit session does not exist, it is automatically created with the specified name.

```
wls:/myserver/serverConfig> edit()
Location changed to edit tree. This is a writeable tree with DomainMBean as the root.
To make changes you will need to start an edit session via startEdit().
For more help, use help('edit')
wls:/wls/edit> edit('mySample')
Edit session mySample does not exist. Creating.

wls:/wls/edit(mySample)>
```

editCustom

Command Category: Browse Commands

Use with WLST: Online

Description

Navigates to the root of custom MBeans that are registered in the Edit MBeanServer. WLST navigates, interrogates, and edits custom MBeans; however, MBeans that are accessed using the `editCustom` command cannot use the `cmo` variable because a stub is not available.

The `editCustom` command is available when WLST is connected to an Administration Server instance.

See Accessing Other WebLogic MBeans and Custom MBeans in *Understanding the WebLogic Scripting Tool*.

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

Syntax

```
editCustom([ObjectNamePattern)
```

Argument	Definition
`ObjectNamePattern`	Optional. A JMX query pattern such as "sip:*". The default value is "*:*" or null.

Example

The following example changes the location to the `editCustom` tree. This is a writable tree with no root.

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

jndi

Command Category: Tree Commands

Use with WLST: Online

Description

Navigates to the JNDI tree for the server to which WLST is currently connected. This read-only tree holds all the elements that are currently bound in JNDI.

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

Syntax

```
jndi()
```

Example

The following example navigates from the runtime MBean hierarchy to the Domain JNDI tree on an Administration Server instance.

```
wls:/myserver/runtime> jndi()
Location changed to jndi tree. This is a read-only tree with No root. For more
help, use help('jndi')
wls:/myserver/jndi> ls()
dr--   ejb
```

```
dr--   javax
dr--   jms
dr--   weblogic
...
```

serverConfig

Command Category: Tree Commands

Use with WLST: Online

Description

Navigates to the last MBean to which you navigated in the configuration MBean hierarchy or to the root of the hierarchy, `DomainMBean`.

This read-only hierarchy stores the configuration MBeans that represent the server to which WLST is currently connected. The MBean attribute values include any command-line overrides that a user specified while starting the server.

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

See Navigating Among MBean Hierarchies in *Understanding the WebLogic Scripting Tool*.

Syntax

```
serverConfig()
```

Example

The following example navigates from the domain runtime MBean hierarchy to the configuration MBean hierarchy on an Administration Server instance.

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

serverRuntime

Command Category: Tree Commands

Use with WLST: Online

Description

Navigates to the last MBean to which you navigated in the runtime MBean hierarchy or to the root of the hierarchy, `ServerRuntimeMBean`. This read-only hierarchy stores the runtime MBeans that represent the server to which WLST is currently connected.

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

Syntax

```
serverRuntime()
```

Example

The following example navigates from the configuration MBean hierarchy to the runtime MBean hierarchy on an Administration Server instance.

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

Store Administration Commands

Use the WLST store administration commands to manage JDBC stores and file stores. Table 2-13 lists and summarizes these commands. For more information about these commands, see Store Administration Using WLST in *Administering the WebLogic Persistent Store*.

Table 2-13 Store Administration Commands

Use this command...	To...	Use with WLST...
closestore	Closes a store.	Offline
compactstore	Compacts and defragments the space occupied by a file store.	Offline
dumpstore	Dumps store contents in human-readable format to an XML file.	Offline
getopenstores	Returns a list of opened stores (for script access).	Offline
getstoreconns	Returns a list of connections int he specified store (for script access).	Offline
liststore	Lists store names, open stores or connections in a store.	Offline
openfilestore	Opens a file store.	Offline
openjdbcstore	Opens a JDBC store.	Offline

closestore

Command Category: Store Administration

Use with WLST: Offline

Description

Closes a previously opened file store or JDBC store. This command returns 1 when successful or a 0 if there is a failure.

Syntax

```
closestore(store)
```

Argument	Definition
store	The name of a previously opened JDBC or file store.

Example

The following example closes a JDBC store called `myJDBCStore`.

```
wls:/offline> closestore('myJDBCStore')
```

compactstore

Command Category: Store Administration

Use with WLST: Offline

Description

Compacts and defragments the space occupied by a file store. This command only works in WLST offline mode and does not work for JDBC stores. This command returns a `1` if successful or a `0` if there is a failure.

Note:

Running file stores are optimized for speed without regard for space. Therefore, running compact will reduce the stores size and utilize its allocated space more efficiently.

Syntax

```
compactstore(dir, [tempdir])
```

Argument	Definition
dir	A directory that contains a file store.
tempdir	Optional. A temporary directory to be used during compacting. This directory should not be under the specified file store directory and must have space sufficient for the file store.

Example

The following example compacts the space occupied by file store files in the `mystores` directory:

```
wls:/offline> compactstore('/mystores', '/tmpmystore.dir')
```

dumpstore

Command Category: Store Administration

Use with WLST: Offline

Description

Dumps store or connection contents in a human-readable format to the specified XML file. The XML file format is the same format used by the diagnostic image of the persistent store. This command returns a `1` if successful or a `0` if there is a failure.

Syntax

```
dumpstore(store, outfile, [conn], [deep])
```

Argument	Definition
store	The name of a previously opened JDBC or file store.
outfile	The name of the XML file to dump the information to, with or without the .xml suffix.
conn	Optional. Store connection name to which the dump should be restricted.
deep	Optional. If `true`, store record data contents are added as a hex dump to the dump output. This argument defaults to `false`.

Example

The following example dumps the contents of `myJDBCStore` to the file mystoredump-out.xml:

```
wls:/offline> dumpstore('myJDBCStore', 'mystoredump-out')
```

getopenstores

Command Category: Store Administration

Use with WLST: Offline

Description

Returns a list of opened stores (for script access). This command returns a `1` if successful or a `0` if there is a failure.

Syntax

```
getopenstores()
```

Example

The following example returns a list of open stores:

```
wls:/offline> getopenstores()
array(java.lang.String,[])
```

getstoreconns

Command Category: Store Administration

Use with WLST: Offline

Description

Returns a list of connections in the specified store (for script access). This command returns a `1` if successful or a `0` if there is a failure.

Syntax

```
getstoreconns(store)
```

Argument	Definition
store	The name of a previously opened JDBC or file store.

Example

The following example returns a list of connections for the JDBC store `myJDBCStore`:

```
wls:/offline> getstoreconns('myJDBCStore')
array(java.lang.String,[])
```

liststore

Command Category: Store Administration

Use with WLST: Offline

Description

Lists storenames, opened stores, or connections (for interactive access). This command returns a `1` if successful or a `0` if there is a failure.

Syntax

```
liststore([store], [dir])
```

Argument	Definition
store	Optional. The name of a previously opened JDBC or file store.
dir	Optional. The directory for which to list available store names.

Example

The following example lists the connections for a JDBC store called `myJDBCStore`:

```
wls:/offline> liststore('myJDBCStore')
```

openfilestore

Command Category: Store Administration

Use with WLST: Offline

Description

Opens an existing file store for further operations. If a file store does not exist, a new one is created in an open state using the `-create` parameter. This command returns a `1` if successful or a `0` if there is a failure.

Syntax

```
openfilestore(store, [dir], [create])
```

Argument	Definition
store	The name of the store to be opened.

Argument	Definition
dir	Optional. The name of a file system directory containing store files. This argument defaults to '`.`'.
create	Optional. If set to `true`, the command creates a non-existing file store. The default value is `false`.

Example

The following example creates a file store called `myJDBCStore`:

```
wls:/offline> openfilestore('myJDBCStore', '', 'true')
```

openjdbcstore

Command Category: Store Administration

Use with WLST: Offline

Description

Opens an existing JDBC store for further operations. If a JDBC store does not exist, a new one is created in an open state. This command returns a `1` if successful or a `0` if there is a failure.

Syntax

```
openjdbcstore(store, [driver], [url], [propfile], [user], [password], [ddl],
tableNamePrefix])
```

Argument	Definition
store	The name of the store to open.
driver	Optional. The name of the JDBC driver class. This argument defaults to null.
url	Optional. URL to connect to the database.
propfile	Optional. JDBC properties file. This argument defaults to null.
user	Optional. The user name for accessing the database. This argument defaults to null.
password	Optional. The password for accessing the database. This argument defaults to null.
ddl	Optional. The name of the DDL file that defines the database table format. This argument defaults to null.
tableNamePrefix	Optional. The prefix for the naming database table. This argument defaults to null.

Example

The following example opens the JDBC store `myJDBCStore`. No DDL file is specified.

```
wls:/offline> openjdbcstore('myJDBCStore', 'oracle.jdbc.OracleDriver',
'jdbc:oracle:thin:@test231:1521:test123', './wlstoreadmin-dump.props',
'dbuser', 'dbpw', '', 'jdbcstoreprefix')
```

WLST Variable Reference

WLST includes a set of variables that you can use in WLST sessions. All variables, listed and described in Table 2-14, are initialized to default values at the start of a user session and are changed according to the user interaction with WLST.

Table 2-14 WLST Variables

Variable	Description	Example
`cmgr`	The `cmgr` variable is set to the `ConfigurationManagerMBean`. You can use this variable to get the current value of any `ConfigurationManagerMBean` attribute.	`wls:/mydomain/edit>` **`cmgr.getCurrentEditor()`** `'weblogic'`
`cmo`	Current Management Object. The `cmo` variable is set to the bean instance to which you navigate using WLST. You can use this variable to perform any `get`, `set`, or `invoke` method on the current bean instance. WLST sets the variable to the current WLST path. For example, when you change to the serverConfig hierarchy, `cmo` is set to DomainMBean. When you change to the serverRuntime hierarchy, `cmo` is set to ServerRuntimeMBean. The variable is available in all WLST hierarchies except custom and jndi.	`wls:/mydomain/edit>` **`cmo.setAdministrationPort(9092)`**
`connected`	Boolean value specifying whether WLST is connected to a running server. WLST sets this variable to `true` when connected to a running server; otherwise, WLST sets it to `false`.	`wls:/mydomain/serverConfig>` **`print connected`** `false`
`domainName`	Name of the WebLogic domain to which WLST is connected.	`wls:/mydomain/serverConfig>` **`print domainName`** `mydomain`
`domainRuntimeService`	`DomainRuntimeServiceMBean` MBean. This variable is available only when WLST is connected to the Administration Server.	`wls:/mydomain/serverConfig>` **`domainRuntimeService.getServerName()`** `'myserver'`
`editService`	`EditServiceMBean` MBean. This variable is available only when WLST is connected to the Administration Server.	`wls:/mydomain/edit>` **`dc = editService.getDomainConfiguration()`**
`exitonerror`	Boolean value specifying whether WLST terminates script execution when it encounters an exception. This variable defaults to `true`, indicating that script execution is terminated when WLST encounters an error. This variable is not applicable when running WLST in interactive mode.	`wls:/mydomain/serverConfig>` **`print exitonerror`** `true`
`idd`	The identity domain of the user who is currently connected to WLST.	`wls:/mydomain/serverConfig>` **`print idd`** `dbUsers`

Table 2-14 (Cont.) WLST Variables

Variable	Description	Example
`isAdminServer`	Boolean value specifying whether WLST is connected to a WebLogic Administration Server instance. WLST sets this variable to `true` if WLST is connected to a WebLogic Administration Server; otherwise, WLST sets it to `false`.	`wls:/mydomain/serverConfig>` **`print isAdminServer`** `true`
`mbs`	`MBeanServerConnection` object that corresponds to the current location in the hierarchy.	`wls:/mydomain/serverConfig>` **`mbs.isRegistered(ObjectName('mydomain: Name=mydomain,Type=Domain'))`**
`recording`	Boolean value specifying whether WLST is recording commands. WLST sets this variable to `true` when the `startRecording` command is entered; otherwise, WLST sets this variable to `false`.	`wls:/mydomain/serverConfig>` **`print recording`** `true`
`runtimeService`	`RuntimeServiceMBean` MBean.	`wls:/mydomain/serverConfig>` **`sr=runtimeService.getServerRuntime()`**
`serverName`	Name of the server to which WLST is connected.	`wls:/mydomain/serverConfig>` **`print serverName`** `myserver`
`typeService`	`TypeServiceMBean` MBean.	`wls:/mydomain/serverConfig>` **`mi=typeService.getMBeanInfo('weblogic. management.configuration.ServerMBean')`**
`username`	Name of user currently connected to WLST.	`wls:/mydomain/serverConfig>` **`print username`** `weblogic`
`version`	Current version of the running server to which WLST is connected.	`wls:/mydomain/serverConfig>` **`print version`** `WebLogic Server 9.0 Thu Aug 31 12:15:50 PST 2005 778899`