# Oracle® Cloud

WLST Command Reference for SOA Suite

12*c* (12.1.3)

**E60658-05**

October 2016

ORACLE

Oracle Cloud WLST Command Reference for SOA Suite, 12*c* (12.1.3)

E60658-05

# Contents

# Preface

This preface describes the document accessibility features and conversions used in this guide—*WLST Command Reference for SOA Suite*.

## 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.

## Conventions

The following text conventions are used in this document:

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

# 1

# Introduction and Roadmap

This section describes the audience for and contents and organization of this guide—*WLST Command Reference for SOA Suite*.



## Differences Between Using this Component in the Cloud and On-Premises Environments

There may be differences between using this component in the cloud and on-premises environments that impact the information described in this guide.

For information about differences, see Differences Between the Cloud and On-Premises Environments and Known Issues for Oracle SOA Cloud Service.

## Document Scope and Audience

This document describes all of the Oracle SOA Suite, Business Process Management, Enterprise Scheduler, and Managed File Transfer WLST commands that are available to use with the WebLogic Scripting Tool (WLST).

> **Note:**
>
> WLST commands for an Oracle Fusion Middleware component are available for use only if the component is installed in the *ORACLE_HOME* directory. So, the SOA Suite WLST commands are available for use only if SOA Suite is installed

This document is written for Oracle SOA Suite administrators and operators, who deploy SOA applications. It is assumed that readers are familiar with Web technologies and the operating system and platform where WebLogic Server and Fusion Middleware products are installed.

## Guide to This Document

This document is organized as follows:

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

- Oracle SOA Suite Custom WLST Commands, provides detailed descriptions for each of the custom WLST commands for SOA Suite.
- Custom WLST Commands , provides detailed descriptions for each of the custom WLST commands for Business Process Management (BPM).
- Oracle Enterprise Scheduler Custom WLST Commands provides detailed descriptions for each of the custom WLST commands for Oracle Enterprise Scheduler (ESS).
- Custom WLST Commands provides detailed descriptions for each of the custom WLST commands for Oracle Managed File Transfer (MFT).

## Related Documentation

For information about how to use the WebLogic Scripting Tool, refer to *Understanding the WebLogic Scripting Tool*.

For information about the other WLST commands and other WebLogic Server management interfaces, see:

- *WLST Command Reference for WebLogic Server* describes the WLST commands for WebLogic Server.
- *WLST Command Reference for Infrastructure Components* describes the WLST commands that are available for Oracle Fusion Middleware core components, including Java Required Files (JRF), Web services, Metadata services (MDS), Application Development Framework (ADF), Dynamic Monitoring Service (DMS), Logging, Diagnostic Framework, User Messaging Service (UMS), and Enterprise Scheduling Service (ESS).
- *Infrastructure Security WLST Command Reference* describes the WLST commands that are available for Oracle Fusion Middleware Infrastructure Security components, including Auditing, SSL, Oracle Identity Federation, Directory Integration Platform, Oracle Access Management (OAM), Oracle Security Token Service, and Oracle Keystore Service.
- *WebCenter WLST Command Reference* describes the WLST commands that are available for WebCenter components, including WebCenter Portal, WebCenter Content, WebCenter Information Rights Management (IRM), and WebCenter Imaging Process Management (IPM).
- "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.
- *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.

# 2

# Oracle SOA Suite Custom WLST Commands

This chapter describes WSLT commands for Oracle SOA Suite. These commands enable you to use WLST to configure SOA composite applications.

> **Note:**
>
> To use these commands, you must invoke WLST from *SOA_HOME*/common/bin.
>
> WLST provides both offline and online modes. Offline commands can be used without connecting to the Administration Server. When you first invoke WLST, you are in offline mode. You need to connect to the Administration server, using the `connect` command, before you can use the online mode commands.

This chapter includes the following sections:



For additional details about deployment, configuration plans, and test suites, see *Developing SOA Applications with Oracle SOA Suite*.

# Overview of WSLT Command Categories

WLST commands are divided into the categories shown in Table 2-1.

***Table 2-1 Oracle SOA Suite Command Categories***

| Command category | Description |
|---|---|
| Deployment Commands | Deploy and undeploy SOA composite applications. |
| SOA Composite Application Management Commands | Start, stop, activate, retire, assign a default revision version, and list deployed SOA composite applications. |
| Configuration Plan Management Commands | Attach, extract, generate, and validate configuration plans for SOA composite applications. |
| Task Validation Commands | Validate human workflow tasks. |
| SOA Composite Application Compilation Commands | Compile SOA composite applications. |
| SOA Composite Application Packaging Commands | Package SOA composite applications into archive files to deploy. |
| SOA Composite Application Test Commands | Test SOA composite applications prior to deployment in a production environment. |
| SOA Composite Application HTTP Client-Based Export and Import Commands | Export and import SOA composite applications based on the HTTP client. |
| SOA Composite Application MBean-Based Export and Import Commands | Export and import SOA composite applications on the server-based composite store MBean (`CompositeStoreMXBean`). |
| SOA Composite Application Partition Management Commands | Logically group different revisions of your SOA composite applications into separate partitions. |
| SOA Composite Application Offline Deployment Management Commands | Manage offline deployments of SOA composite applications and shared data. |

# Deployment Commands

Use the deployment commands, listed in Table 2-2, to deploy and undeploy SOA composite applications.

***Table 2-2 Deployment Commands for WLST Configuration***

| Use this command... | To... | Use with WLST... |
|---|---|---|
| sca_deployComposite | Deploy a SOA composite application. | Offline |
| sca_undeployComposite | Undeploy a SOA composite application. | Offline |

## sca_deployComposite

Command Category: Deployment Commands

Use with WLST: Offline

### Description

Deploys a SOA composite application to the Oracle WebLogic Server. This command does *not* package the artifact files of the application for deployment. See SOA Composite Application Packaging Commands for instructions on packaging a SOA composite application.

### Syntax

```
sca_deployComposite(serverURL, sarLocation, [overwrite], [user], [password],
[forceDefault], [configplan], [partition] [keepInstancesOnRedeploy])
```

| Argument | Definition |
|---|---|
| *serverURL* | URL of the server that hosts the SOA Infrastructure application (for example, `http://myhost10:7001`). |
| *sarLocation* | Absolute path to one the following:<br>• SOA archive (SAR) file.<br>A SAR file is a special JAR file that requires a prefix of `sca_` (for example, `sca_HelloWorld_rev1.0.jar`). The SAR file can be deployed with the deployment commands (such as `sca_deployComposite()`), but a regular `.jar` file is not treated as a special SAR file.<br>• ZIP file that includes multiple SARs, metadata archives (MARs), or both. |
| *overwrite* | Optional. Indicates whether to overwrite an existing SOA composite application.<br>• `false` (default): Does not overwrite the application.<br>• `true`: Overwrites the application. |
| *user* | Optional. User name to access the composite deployer servlet when basic authentication is configured. |

| Argument | Definition |
|---|---|
| *password* | Optional. Password to access the composite deployer servlet when basic authentication is configured. |
| *forceDefault* | Optional. Indicates whether to set the new composite as the default.<br>• `true` (default): Makes it the default composite.<br>• `false`: Does not make it the default composite. |
| *configplan* | Optional. Absolute path of a configuration plan to be applied to a specified SAR file or to all SAR files included in the ZIP file. |
| *partition* | Optional. The name of the partition in which to deploy the SOA composite application. The default value is `default`. If you do not specify a partition, the composite is automatically deployed into the `default` partition. |
| *keepInstancesOnRedeploy* | Optional. Specifies whether to keep the instances on redeploy. The default value is false. This argument is used when BPM is installed. |

**Note:**

Human workflow artifacts such as task mapped attributes (previously known as flex field mappings) and rules (such as vacation rules) are defined based on the namespace of the task definition. Therefore, the following issues are true when the same SOA composite application with a human workflow task is deployed into multiple partitions:

- For the same task definition type, mapped attributes defined in one partition are visible in another partition.
- Rules defined on a task definition in one partition can apply to the same definition in another partition.

## Examples

The following example deploys the `HelloWorld` application.

```
wls:/mydomain/ServerConfig> sca_deployComposite("http://myhost10:7001",
"/tmp/sca_HelloWorld_rev1.0.jar")
```

The following example deploys the `HelloWorld` application as the default version.

```
wls:/mydomain/ServerConfig> sca_deployComposite("http://myhost10:7001",
"/tmp/sca_HelloWorld_rev1.0.jar", true)
```

The following example deploys the `HelloWorld` application with a required user name when basic authentication is configured. You are then prompted to provide the password for this user name.

```
wls:/mydomain/ServerConfig> sca_deployComposite("http://myhost10:7001",
"/tmp/sca_HelloWorld_rev1.0.jar", user="weblogic")
Password:
```

The following example deploys the `HelloWorld` application and applies the configuration plan named `deployplan.xml`.

```
wls:/mydomain/ServerConfig> sca_deployComposite("http://myhost10:7001",
"/tmp/sca_HelloWorld_rev1.0.jar", forceDefault=false,
configplan="/tmp/deployplan.xml")
```

The following example deploys the `HelloWorld` ZIP file, which can include multiple SARs, MARs, or both.

```
wls:/mydomain/ServerConfig> sca_deployComposite("http://myhost:7001",
"/tmp/HelloWorld.zip")
```

The following example deploys the `HelloWorld` application to the `myPartition` partition.

```
wls:/mydomain/ServerConfig> sca_deployComposite("http://stadp10:7001",
"/tmp/sca_HelloWorld_rev1.0.jar", partition="myPartition")
```

## sca_undeployComposite

Command Category: Deployment Commands

Use with WLST: Offline

### Description

Undeploys a currently deployed SOA composite application.

### Syntax

```
sca_undeployComposite(serverURL, compositeName, revision, [user], [password],
[partition])
```

| Argument | Definition |
|---|---|
| *serverURL* | URL of the server that hosts the SOA Infrastructure application (for example, `http://myhost10:7001`). |
| *compositeName* | Name of the SOA composite application. |
| *revision* | Revision ID of the SOA composite application. |
| *user* | Optional. User name to access the composite deployer servlet when basic authentication is configured. |
| *password* | Optional. Password to access the composite deployer servlet when basic authentication is configured. |
| *partition* | Optional. The name of the partition in which the SOA composite application is located. The default value is `default`. If you do not specify a partition, the `default` partition is searched for the SOA composite application. However, no other partitions are searched. |

### Examples

The following example undeploys the `HelloWorld` application.

```
wls:/mydomain/ServerConfig> sca_undeployComposite("http://myhost10:7001",
"HelloWorld", "1.0")
```

The following example undeploys the `HelloWorld` application with a required user name when basic authentication is configured. You are then prompted to provide the password for this user name.

```
wls:/mydomain/ServerConfig> sca_undeployComposite("http://myhost10:7001",
"HelloWorld", "1.0", user="weblogic")
Password:
```

The following example undeploys the `HelloWorld` application in the `myPartition` partition.

```
wls:/mydomain/ServerConfig> sca_undeployComposite("http://stadp10:7001",
 "HelloWorld", "1.0", partition='myPartition')
```

# SOA Composite Application Management Commands

Use the management commands, listed in Table 2-3, to start, stop, activate, retire, assign a default revision version, and list deployed SOA composite applications.

***Table 2-3 SOA Composite Application Management Commands for WLST Configuration***

| Use this command... | To... | Use with WLST... |
|---|---|---|
| sca_startComposite | Start a previously stopped SOA composite application. | Offline |
| sca_stopComposite | Stop a SOA composite application. | Offline |
| sca_activateComposite | Activate a previously retired SOA composite application. | Offline |
| sca_retireComposite | Retire a SOA composite application. | Offline |
| sca_assignDefaultComposite | Assign the default revision version to a SOA composite application. | Offline |
| sca_listDeployedComposites | List the deployed SOA composite applications. | Offline |

## sca_startComposite

Command Category: Application Management Commands

Use with WLST: Offline

### Description

Starts a previously stopped SOA composite application.

### Syntax

```
sca_startComposite(host, port, user, password, compositeName, revision, [label],
[partition])
```

| Argument | Definition |
|---|---|
| *host* | Hostname of the Oracle WebLogic Server (for example, `myhost`). |

| Argument | Definition |
|---|---|
| *port* | Port of the Oracle WebLogic Server (for example, 7001). |
| *user* | User name for connecting to the running server to get MBean information (for example, weblogic). |
| *password* | Password for the user name. |
| *compositeName* | Name of the SOA composite application. |
| *revision* | Revision of the SOA composite application. |
| *label* | Optional. Label of the SOA composite application. The label identifies the metadata service (MDS) artifacts associated with the application. If the label is not specified, the system finds the latest one. |
| *partition* | Optional. The name of the partition in which the SOA composite application is located. The default value is default. If you do not specify a partition, the default partition is searched for the SOA composite application. However, no other partitions are searched. |

### Example

The following example starts revision 1.0 of the HelloWorld application.

```
wls:/mydomain/ServerConfig> sca_startComposite("myhost", "7001", "weblogic",
"welcome1", "HelloWorld", "1.0")
```

The following example starts revision 1.0 of the HelloWorld application in the partition myPartition.

```
wls:/mydomain/ServerConfig> sca_startComposite("stadp10", "7001", "weblogic",
 "weblogic", "HelloWorld", "1.0", partition="myPartition")
```

## sca_stopComposite

Command Category: Application Management Commands

Use with WLST: Offline

### Description

Stops a currently running SOA composite application.

### Syntax

```
sca_stopComposite(host, port, user, password, compositeName, revision, [label],
 [partition])
```

| Argument | Definition |
|---|---|
| *host* | Hostname of the Oracle WebLogic Server (for example, myhost). |
| *port* | Port of the Oracle WebLogic Server (for example, 7001). |
| *user* | User name for connecting to the running server to get MBean information (for example, weblogic). |

| Argument | Definition |
|---|---|
| *password* | Password for the user name. |
| *compositeName* | Name of the SOA composite application. |
| *revision* | Revision of the SOA composite application. |
| *label* | Optional. Label of the SOA composite application. The label identifies the MDS artifacts associated with the application. If the label is not specified, the system finds the latest one. |
| *partition* | Optional. The name of the partition in which the SOA composite application is located. The default value is `default`. If you do not specify a partition, the `default` partition is searched for the SOA composite application. However, no other partitions are searched. |

### Example

The following example stops revision `1.0` of the `HelloWorld` application.

```
wls:/mydomain/ServerConfig> sca_stopComposite("myhost", "7001", "weblogic",
"welcome1", "HelloWorld", "1.0")
```

The following example stops revision `1.0` of the `HelloWorld` application in the partition `myPartition`.

```
wls:/mydomain/ServerConfig> sca_stopComposite("stadp10", "7001", "weblogic",
 "weblogic", "HelloWorld", "1.0", partition="myPartition")
```

## sca_activateComposite

Command Category: Application Management Commands

Use with WLST: Offline

### Description

Activates a retired SOA composite application. You can then create new instances.

### Syntax

```
sca_activateComposite(host, port, user, password, compositeName, revision, [label],
[partition])
```

| Argument | Definition |
|---|---|
| *host* | Hostname of the Oracle WebLogic Server (for example, `myhost`). |
| *port* | Port of the Oracle WebLogic Server (for example, `7001`). |
| *user* | User name for connecting to the running server to get MBean information (for example, `weblogic`). |
| *password* | Password for the user name. |
| *compositeName* | Name of the SOA composite application. |
| *revision* | Revision of the SOA composite application. |

| Argument | Definition |
|---|---|
| *label* | Optional. Label of the SOA composite application. The label identifies the MDS artifacts associated with the application. If the label is not specified, the system finds the latest one. |
| *partition* | Optional. The name of the partition in which the SOA composite application is located. The default value is `default`. If you do not specify a partition, the `default` partition is searched for the SOA composite application. However, no other partitions are searched. |

### Example

The following example activates revision `1.0` of the `HelloWorld` application.

```
wls:/mydomain/ServerConfig> sca_activateComposite("myhost", "7001", "weblogic",
"welcome1", "HelloWorld", "1.0")
```

The following example activates revision `1.0` of the `HelloWorld` application in the partition `myPartition`.

```
wls:/mydomain/ServerConfig> sca_activateComposite("stadp10", "7001", "weblogic",
 "weblogic", "HelloWorld", "1.0", partition="myPartition")
```

## sca_retireComposite

Command Category: Application Management Commands

Use with WLST: Offline

### Description

Retires a SOA composite application. If the process life cycle is retired, you cannot create a new instance. Existing instances are allowed to complete normally.

### Syntax

```
sca_retireComposite(host, port, user, password, compositeName, revision, [label],
[partition])
```

| Argument | Definition |
|---|---|
| *host* | Hostname of the Oracle WebLogic Server (for example, `myhost`). |
| *port* | Port of the Oracle WebLogic Server (for example, `7001`). |
| *user* | User name for connecting to the running server to get MBean information (for example, `weblogic`). |
| *password* | Password for the user name. |
| *compositeName* | Name of the SOA composite application. |
| *revision* | Revision of the SOA composite application. |
| *label* | Optional. Label of the SOA composite application. The label identifies the MDS artifacts associated with the application. If the label is not specified, the system finds the latest one. |

| Argument | Definition |
|---|---|
| *partition* | Optional. The name of the partition in which the SOA composite application is located. The default value is `default`. If you do not specify a partition, the `default` partition is searched for the SOA composite application. However, no other partitions are searched. |

### Example

The following example retires revision `1.0` of the `HelloWorld` application.

```
wls:/mydomain/ServerConfig> sca_retireComposite("myhost", "7001", "weblogic",
"welcome1", "HelloWorld", "1.0")
```

The following example retires revision `1.0` of the `HelloWorld` application in the partition `myPartition`.

```
wls:/mydomain/ServerConfig> sca_retireComposite("stadp10", "7001", "weblogic",
 "weblogic", "HelloWorld", "1.0", partition="myPartition")
```

## sca_assignDefaultComposite

Command Category: Application Management Commands

Use with WLST: Offline

### Description

Sets a SOA composite application revision as the default version. This revision is instantiated when a new request comes in.

### Syntax

```
sca_assignDefaultComposite(host, port, user, password, compositeName, revision,
[partition])
```

| Argument | Definition |
|---|---|
| *host* | Hostname of the Oracle WebLogic Server (for example, `myhost`). |
| *port* | Port of the Oracle WebLogic Server (for example, `7001`). |
| *user* | User name for connecting to the running server to get MBean information (for example, `weblogic`). |
| *password* | Password for the user name. |
| *compositeName* | Name of the SOA composite application. |
| *revision* | Revision of the SOA composite application. |
| *partition* | Optional. The name of the partition in which the SOA composite application is located. The default value is `default`. If you do not specify a partition, the `default` partition is searched for the SOA composite application. However, no other partitions are searched. |

### Example

The following example sets revision `1.0` of the `HelloWorld` application as the default version.

```
wls:/mydomain/ServerConfig> sca_assignDefaultComposite("myhost", "7001",
"weblogic", "welcome1", "HelloWorld", "1.0")
```

The following example sets revision `1.0` of the `HelloWorld` application located in the partition `myPartition` as the default version.

```
wls:/mydomain/ServerConfig> sca_assignDefaultComposite("stadp10", "7001",
 "weblogic", "weblogic", "HelloWorld", "1.0", partition="myPartition")
```

## sca_listDeployedComposites

Command Category: Application Management Commands

Use with WLST: Offline

### Description

Lists all SOA composite applications deployed to the SOA platform.

### Syntax

```
sca_listDeployedComposites(host, port, user, password)
```

| Argument | Definition |
|---|---|
| *host* | Hostname of the Oracle WebLogic Server (for example, `myhost`). |
| *port* | Port of the Oracle WebLogic Server (for example, `7001`). |
| *user* | User name for connecting to the running server to get MBean information (for example, `weblogic`). |
| *password* | Password for the user name. |

### Example

The following example lists all the deployed SOA composite applications on the server `myhost`.

```
wls:/mydomain/ServerConfig> sca_listDeployedComposites('myhost', '7001',
'weblogic', 'welcome1')
```

# Configuration Plan Management Commands

Use the configuration plan management commands, listed in Table 2-4, to attach, extract, generate, and validate configuration plans for SOA composite applications.

***Table 2-4 Configuration Plan Management Commands for WLST Configuration***

| Use this command... | To... | Use with WLST... |
|---|---|---|
| sca_attachPlan | Attach the configuration plan file to the SOA composite application JAR file. | Offline |

*Table 2-4 (Cont.) Configuration Plan Management Commands for WLST Configuration*

| Use this command... | To... | Use with WLST... |
|---|---|---|
| sca_extractPlan | Extract a configuration plan packaged with the JAR file for editing. | Offline |
| sca_generatePlan | Generate a configuration plan for editing. | Offline |
| sca_validatePlan | Validate the configuration plan. | Offline |

## sca_attachPlan

Command Category: Configuration Plan Management Commands

Use with WLST: Offline

### Description

Attaches the configuration plan file to the SOA composite application file. If a plan already exists in the file, it is not overwritten by default.

### Syntax

```
sca_attachPlan(sar, configPlan, [overwrite], [verbose])
```

| Argument | Definition |
|---|---|
| *sar* | Absolute path of the SAR file. |
| *configPlan* | Absolute path of the configuration plan file. |
| *overwrite* | Optional. Indicates whether to overwrite an existing configuration plan in the SAR file.<br>• `false` (default): Does not overwrite the plan.<br>• `true`: Overwrites the plan. |
| *verbose* | Optional. Indicates whether to print more information about the configuration plan attachment.<br>• `true` (default): Prints more information.<br>• `false`: Does not print more information. |

### Examples

The following example attaches the `configplan.xml` configuration plan file to the `HelloWorld` application.

```
wls:/mydomain/ServerConfig> sca_attachPlan("/tmp/sca_HelloWorld_rev1.0.jar",
"/tmp/configplan.xml")
```

The following example overwrites the existing configuration plan with `configplan.xml` file in the `HelloWorld` application.

```
wls:/mydomain/ServerConfig> sca_attachPlan("/tmp/sca_HelloWorld_rev1.0.jar", "/tmp/
configplan.xml", overwrite=true)
```

## sca_extractPlan

Command Category: Configuration Plan Management Commands

Use with WLST: Offline

### Description

Extracts a configuration plan packaged with the SOA composite application file for editing. This is an optional step. If no plan exists, this is the same as creating a new file with `sca_generatePlan`.

### Syntax

```
sca_extractPlan(sar, configPlan, [overwrite], [verbose])
```

| Argument | Definition |
|---|---|
| *sar* | Absolute path of a SAR file. |
| *configPlan* | Absolute path of a configuration plan file to which to be extracted. |
| *overwrite* | Optional. Indicates whether to overwrite the configuration file specified by *configPlan*, if it exists.<br>• `false` (default): Does not overwrite the plan.<br>• `true`: Overwrites the plan. |
| *verbose* | Optional. Indicates whether to print more information about configuration plan extraction.<br>• `true` (default): Prints more information.<br>• `false`: Does not print more information. |

### Example

The following example extracts the `configplan.xml` file for editing from the `HelloWorld` application.

```
wls:/mydomain/ServerConfig> sca_extractPlan("/tmp/sca_HelloWorld_rev1.0.jar",
"/tmp/configplan.xml")
```

The following example extracts the `configplan.xml` file for editing from the `HelloWorld` application. This command also overwrites the existing plan.

```
wls:/mydomain/ServerConfig> sca_extractPlan("/tmp/sca_HelloWorld_rev1.0.jar",
"/tmp/configplan.xml", overwrite=true)
```

## sca_generatePlan

Command Category: Configuration Plan Management Commands

Use with WLST: Offline

### Description

Generates a configuration plan for editing.

### Syntax

```
sca_generatePlan(configPlan, sar, composite, [overwrite], [verbose])
```

| Argument | Definition |
|---|---|
| *configPlan* | Absolute path of the configuration plan file to be generated.<br>Either *configPlan* or *sar* is required. You cannot specify both. |
| *sar* | Absolute path of the SAR file.<br>Either *configPlan* or *sar* is required. You cannot specify both. |
| *composite* | Absolute path of the `composite.xml` file in the expanded (unzipped) SAR directory. |
| *overwrite* | Optional. Indicates whether to overwrite an existing configuration plan file:<br>• `false` (default): Does not overwrite the plan.<br>• `true`: Overwrites the plan. |
| *verbose* | Indicates whether to print more information about plan generation:<br>• `true` (default): Prints more information.<br>• `false`: Does not print more information. |

### Examples

The following example generates the `myplan.xml` configuration plan file for the `HelloWorld` application.

```
wls:/mydomain/ServerConfig> sca_generatePlan("/tmp/myplan.xml",
sar="/tmp/sca_HelloWorld_rev1.0.jar")
```

The following example generates the `myplan2.xml` configuration plan file for the `HelloWorld` application. The `myplan2.xml` file overwrites the existing plan.

```
wls:/mydomain/ServerConfig> sca_generatePlan("/tmp/myplan2.xml",
composite="/tmp/HelloWorld_rev1.0/composite.xml", overwrite=true)
```

## sca_validatePlan

Command Category: Configuration Plan Management Commands

Use with WLST: Offline

### Description

Validates the configuration plan. This command identifies all search and replacement changes to be made during deployment. Use this option for debugging only.

### Syntax

```
sca_validatePlan(reportFile, configPlan, [sar], [composite], [overwrite], [verbose])
```

| Argument | Definition |
|---|---|
| *reportFile* | Absolute path of the report file to be generated. Validation results are written to this file. |
| *configPlan* | Absolute path of the configuration plan file. |
| *sar* | Optional. The absolute path of the SAR file. |

| Argument | Definition |
|---|---|
| *composite* | Optional. The absolute path of the `composite.xml` file in the expanded (unzipped) SAR directory. |
| *overwrite* | Optional. Indicates whether to overwrite an existing configuration plan file:<br>• `false` (default): Does not overwrite the plan.<br>• `true`: Overwrites the plan. |
| *verbose* | Optional. Indicates whether to print more information about configuration plan validation.<br>• `true` (default): Prints more information.<br>• `false`: Does not print more information. |

### Examples

The following example validates the `configplan.xml` configuration plan file for the `HelloWorld` application.

```
wls:/mydomain/ServerConfig> sca_validatePlan("/tmp/myreport.xml",
"/tmp/configplan.xml", sar="/tmp/sca_HelloWorld_rev1.0.jar")
```

The following example validates the `configplan.xml` configuration plan file for the `HelloWorld` application. The `configplan.xml` plan overwrites the existing plan.

```
wls:/mydomain/ServerConfig> sca_validatePlan("/tmp/myreport.xml",
"/tmp/configplan.xml",composite="/tmp/HelloWorld_rev1.0/composite.xml",
overwrite=true)
```

## Task Validation Commands

Use the task validation command, listed in Table 2-5, to validate human workflow tasks.

***Table 2-5 Task Validation Command for WLST Configuration***

| Use this command... | To... | Use with WLST... |
|---|---|---|
| sca_validateTask | Validate a human workflow task. | Offline |

## sca_validateTask

Command Category: Task Validation Commands

Use with WLST: Offline

### Description

Validates a human workflow task contained in the `.task` file that you created when designing a human task in the Human Task Editor.

### Syntax

```
sca_validateTask(taskFile, outXml, [displayLevel])
```

| Argument | Definition |
|---|---|
| *taskFile* | Absolute path to the task definition file (`.task`). |
| *outXml* | Absolute path to an output XML file. |
| *displayLevel* | Optional. The level of information to display. The values can be 1, 2, or 3. The default value is `1`. |

### Example

The following example validates the `WFTaskDefinition.task` file of the human task.

```
wls:/mydomain/ServerConfig> sca_validateTask("/tmp/WFTaskDefinition.task",
"/tmp/out.xml", displayLevel=2)
```

# SOA Composite Application Compilation Commands

Use the compilation commands, listed in Table 2-6, to compile SOA composite applications.

***Table 2-6 SOA Composite Application Compilation Commands for WLST Configuration***

| Use this command... | To... | Use with WLST... |
|---|---|---|
| sca_setProp | Set JVM system properties. | Offline |
| sca_compile | Compile a SOA composite application. | Offline |

## sca_setProp

Command Category: Application Compilation Commands

Use with WLST: Offline

### Description

Sets JVM system properties. This command can also set secure socket layer (SSL) system properties before using `sca_deployComposite` and `sca_undeployComposite` over SSL.

### Syntax

```
sca_setProp(propName, propValue)
```

| Argument | Definition |
|---|---|
| *propName* | Property name. |
| *propValue* | Property value. |

### Example

The following example sets the property name and property value.

```
wls:/mydomain/ServerConfig> sca_setProp("oracle.home",
"/scratch/myusername/beahome/AS11gR1SOA")
```

## sca_compile

Command Category: Application Compilation Commands

Use with WLST: Offline

### Description

Compiles a SOA composite application.

**Note:**

The `sca_compile` command requires the `oracle.home` property to find the `ant-sca-compile.xml` script. This must be set once. You can use the `scac_setProp` command or the `oracleHome` property to set a value.

### Syntax

```
sca_compile(composite, [outXml], [error], [appHome], [displayLevel], [oracleHome])
```

| Argument | Definition |
|---|---|
| *composite* | Absolute path of a composite file in the expanded (unzipped) SAR directory. |
| *outXml* | Optional. Absolute path of an output XML file. |
| *error* | Optional. Absolute path of an error file. |
| *appHome* | Optional. Absolute path of the application home directory. This property is required if you have shared data. |
| *displayLevel* | Optional. The level of information to display. The default value is `1`. |
| *oracleHome* | Optional. The `oracle.home` property. |
| *reportSchemaValidationErrors* | Optional. Indicates whether schema validation error is reported. The default value is false. |

### Examples

The following example compiles the `FirstComposite` application.

```
wls:/mydomain/ServerConfig> sca_compile("/tmp/FirstComposite_
rev1.0/composite.xml", displayLevel=2)
```

The following example compiles the `FirstComposite` application and captures details in the `myout.xml` file. The `error.out` file captures any errors.

```
wls:/mydomain/ServerConfig> sca_compile("/tmp/FirstComposite_
rev1.0/composite.xml", outXml="/tmp/myout.xml", error="error.out")
```

The following example compiles the `FirstComposite` application. The `oracleHome` property is set to find the `ant-sca-compile.xml` script.

```
wls:/mydomain/ServerConfig> sca_compile("/tmp/FirstComposite_
rev1.0/composite.xml", displayLevel=2,
oracleHome="/scratch/myusername/beahome/AS11gR1SOA")
```

# SOA Composite Application Packaging Commands

Use the packaging command, listed in Table 2-7, to package SOA composite applications into a composite SAR file.

***Table 2-7 SOA Composite Application Packaging Command for WLST Configuration***

| Use this command... | To... | Use with WLST... |
|---|---|---|
| sca_package | Package the SOA composite application files into a composite SAR file. | Offline |

## sca_package

Command Category: Application Packaging Commands

Use with WLST: Offline

### Description

Packages the SOA composite application files into a composite SAR file. This command performs the following operations:

- Calls `sca_compile` to compile the composite artifacts in `${compositeDir}`.
- Calls `javac` to compile any source code under `${compositeDir}/src`.
- Replaces the revision in `${compositeDir}/composite.xml`.
- Packages the artifacts to create `sca_${compositeName}_rev${revision}.jar` in `${compositeDir}/deploy`.

**Note:**

The `sca_package` command requires `oracle.home` to find the `ant-sca-package.xml` script. This must be set once. You can use the `scac_setProp` command or `oracleHome` property to set this property.

### Syntax

```
sca_package(compositeDir, compositeName, revision, [appHome], [oracleHome])
```

| Argument | Definition |
|---|---|
| *compositeDir* | Absolute path of a directory that contains composite artifacts. |
| *compositeName* | Name of the composite. |
| *revision* | Revision ID of the composite. |

| Argument | Definition |
|---|---|
| *appHome* | Optional. Absolute path of the application home directory. This property is required if you have shared data. |
| *oracleHome* | Optional. The `oracle.home` property. |

### Examples

The following example packages the `OrderBookingComposite` application. The `appHome` property is set because this application uses shared data.

```
wls:/mydomain/ServerConfig> sca_package("/tmp/app_data/OrderBookingComposite",
"OrderBookingComposite", "1.0", appHome="/tmp/app_data")
```

The following example packages the `HelloSOAComposite` application.

```
wls:/mydomain/ServerConfig> sca_package
("/tmp/HelloSOAApplication/HelloSOAComposite", "HelloSOAComposite", "1.0")
```

The following example packages the `HelloSOAComposite` application. The `oracleHome` property is set to find the `ant-sca-compile.xml` script.

```
wls:/mydomain/ServerConfig> sca_package
("/tmp/HelloSOAApplication/HelloSOAComposite", "HelloSOAComposite", "1.0",
oracleHome="/scratch/myusername/beahome/AS11gR1SOA")
```

## SOA Composite Application Test Commands

Use the SOA composite application test command, listed in Table 2-8, to test a SOA composite applications.

***Table 2-8 SOA Composite Application Test Command for WLST Configuration***

| Use this command... | To... | Use with WLST... |
|---|---|---|
| sca_test | Test deployed SOA composite applications. | Offline |

## sca_test

Command Category: Application Test Commands

Use with WLST: Offline

### Description

Tests deployed SOA composite applications prior to deployment in a production environment. You create suites of tests in Oracle JDeveloper. The `sca_test` command calls `ant-sca-test.xml`.

### Syntax

```
sca_test('compositeName', 'revision', 'testsuiteName', 'jndiPropFile',
[oracleHome='oracleHome'], [javaHome='javaHome'])
```

| Argument | Definition |
|---|---|
| *compositeName* | Name of the SOA composite application. |
| *revision* | Revision ID of the SOA composite application. |
| *testsuiteName* | Name of the test suite. |
| *jndiPropFile* | Absolute path to the JNDI property file. |
| *oracleHome* | Optional. The `oracle.home` system property. |
| *javaHome* | Optional. The `java.passed.home` system property. |

### Examples

The following example runs the `OrderBookingMainTestsuite` test suite.

```
wls:/mydomain/ServerConfig> sca_test('OrderBookingComposite', '1.0',
 'OrderBookingMainTestsuite', '/tmp/tmp-jndi.properties',
 oracleHome='/scratch/<user>/beahome/AS11gR1SOA/',
 javaHome='/scratch/<user>/beahome/jdk160_05')
```

# SOA Composite Application HTTP Client-Based Export and Import Commands

Use the SOA composite application commands, listed in Table 2-9, to export and import SOA composite applications based on the HTTP client. The SOA Infrastructure must be running to use these commands.

***Table 2-9 SOA Composite Application Export and Import Commands for WLST Configuration***

| Use this command... | To... | Use with WLST... |
|---|---|---|
| sca_exportComposite | Export a SOA composite application into a SAR file. | Offline |
| sca_exportUpdates | Export postdeployment changes of a SOA composite application into a JAR file. | Offline |
| sca_importUpdates | Import postdeployment changes of a SOA composite application. | Offline |
| sca_exportSharedData | Export shared data of a given pattern into a JAR file. | Offline |
| sca_removeSharedData | Removes a top-level shared data folder. | Offline |

## sca_exportComposite

Command Category: Application Export and Import Commands

Use with WLST: Offline

### Description

Exports a SOA composite application into a SAR file.

### Syntax

```
sca_exportComposite(serverURL, updateType, sarFile, compositeName, revision,
[user], [password], [partition])
```

| Argument | Definition |
|---|---|
| *serverURL* | URL of the server that hosts the SOA Infrastructure application (for example, `http://stabc:8001`). |
| *updateType* | Type of postdeployment changes to be exported:<br>• `all`: Includes all postdeployment changes.<br>• `property`: Includes only property postdeployment changes (binding component properties, composite properties such as audit level settings and payload validation status, and policy attachments).<br>• `runtime`: Includes only runtime (rules dictionary and domain value maps (DVMs)) and metadata postdeployment changes.<br>• `none`: Exports the original composite without any postdeployment changes (including property changes and runtime changes). |
| *sarFile* | Absolute path of a SAR file to generate (a `.jar` file that begins with `sca_`). |
| *compositeName* | Name of the composite to export. |
| *revision* | Revision of the composite to export. |
| *user* | Optional. The user name for accessing the server when basic configuration is configured. Use the following syntax for this argument:<br>`user='username'` |
| *password* | Optional. The password for accessing the server when basic configuration is configured. Use the following syntax for this argument:<br>`password='password'` |
| *partition* | Optional. The name of the partition in which the SOA composite application is located. The default value is `default`. |

### Examples

The following example exports the composite without including any postdeployment changes.

```
wls:/offline/mydomain/ServerConfig> sca_exportComposite('http://stabc:8001',
'none', '/tmp/sca_HelloWorld_rev1.0.jar', 'HelloWorld', '1.0')
```

The following example exports a composite with all postdeployment updates.

```
wls:/offline/mydomain/ServerConfig> sca_exportComposite('http://stabc:8001',
'all', '/tmp/sca_HelloWorld_ rev1.0-all.jar', 'HelloWorld', '1.0')
```

The following example exports a composite with property postdeployment updates.

```
wls:/offline/mydomain/ServerConfig> sca_exportComposite('http://stabc:8001',
'property', '/tmp/sca_HelloWorld_ rev1.0-prop.jar', 'HelloWorld', '1.0')
```

The following example exports a composite with runtime/metadata postdeployment updates.

```
wls:/offline/mydomain/ServerConfig> sca_exportComposite('http://stabc:8001',
'runtime', '/tmp/sca_HelloWorld_ rev1.0-runtime.jar', 'HelloWorld', '1.0')
```

The following example exports a composite in the `myPartition` partition without including any postdeployment updates:

```
wls:/offline/mydomain/ServerConfig> sca_exportComposite('http://stabc:8001',
 'none', '/tmp/sca_HelloWorld_rev1.0.jar', 'HelloWorld', '1.0',
 partition='myPartition')
```

## sca_exportUpdates

Command Category: Application Export and Import Commands

Use with WLST: Offline

### Description

Exports postdeployment changes of a SOA composite application into a JAR file.

### Syntax

```
sca_exportUpdates(serverURL, updateType, jarFile, compositeName, revision,
[user], [password], [partition])
```

| Argument | Definition |
|---|---|
| *serverURL* | URL of the server that hosts the SOA Infrastructure application (for example, `http://stabc:8001`). |
| *updateType* | The type of postdeployment changes to be exported.<br>• `all`: Includes all postdeployment changes.<br>• `property`: Includes only property postdeployment changes (binding component properties, composite properties such as audit level settings and payload validation status, and policy attachments).<br>• `runtime`: Includes only runtime (rules dictionary and domain value maps (DVMs)) and metadata postdeployment changes. |
| *jarFile* | Absolute path of a JAR file to generate. `sca_exportUpdates()` creates a regular `.jar` file that cannot be imported using regular deployment commands. It must be imported by using `sca_importUpdates()`. |
| *compositeName* | Name of the composite to export. |
| *revision* | Revision of the composite to export. |
| *user* | Optional. The user name for accessing the server when basic configuration is configured. Use the following syntax for this argument:<br>`user='`*username*`'` |
| *password* | Optional. The password for accessing the server when basic configuration is configured. Use the following syntax for this argument:<br>`password='`*password*`'` |

| Argument | Definition |
|---|---|
| *partition* | Optional. The name of the partition in which the SOA composite application is located. The default value is `default`. |

### Examples

The following example exports all postdeployment updates.

```
wls:/offline/mydomain/ServerConfig>  sca_exportUpdates('http://stabc:8001', 'all',
'/tmp/all-HelloWorld_rev1.0.jar','HelloWorld', '1.0')
```

The following example exports property postdeployment updates.

```
wls:/offline/mydomain/ServerConfig> sca_exportUpdates('http://stabc:8001',
'property','/tmp/prop-HelloWorld_ rev1.0.jar', 'HelloWorld', '1.0')
```

The following example exports runtime/metadata postdeployment updates.

```
wls:/offline/mydomain/ServerConfig> sca_exportUpdates('http://stabc:8001',
'runtime','/tmp/runtime-HelloWorld_ rev1.0.jar', 'HelloWorld', '1.0')
```

The following example exports postdeployment changes of a composite in the partition `myPartition` into a JAR file.

```
wls:/offline/mydomain/ServerConfig> sca_exportUpdates('http://stabc:8001',
'runtime', '/tmp/runtime-HelloWorld_rev1.0.jar', 'HelloWorld', '1.0',
partition='myPartition')
```

## sca_importUpdates

Command Category: Application Export and Import Commands

Use with WLST: Offline

### Description

Imports postdeployment changes of a SOA composite application.

### Syntax

```
sca_importUpdates(serverURL, jarFile, compositeName, revision, [user],
[password])
```

| Argument | Definition |
|---|---|
| *serverURL* | URL of the server that hosts the SOA Infrastructure application (for example, `http://stabc:8001`). |
| *jarFile* | Absolute path of a JAR file that contains postdeployment changes. |
| *compositeName* | Name of the composite to which the postdeployment changes are imported. |
| *revision* | Revision of the composite to which the postdeployment changes are imported. |

| Argument | Definition |
|---|---|
| *user* | Optional. The user name for accessing the server when basic configuration is configured. Use the following syntax for this argument:<br>`user='username'` |
| *password* | Optional. The password for accessing the server when basic configuration is configured. Use the following syntax for this argument:<br>`password='password'` |
| *partition* | Optional. The name of the partition in which the SOA composite application is located. The default value is `default`. |

### Examples

The following example imports postdeployment changes of a SOA composite application.

```
wls:/offline/mydomain/ServerConfig> sca_importUpdates('http://stabc:8001',
'/tmp/all-HelloWorld_rev1.0.jar', 'HelloWorld', '1.0')
```

The following example imports postdeployment changes of a composite in the partition `myPartition`.

```
wls:/offline/mydomain/ServerConfig> sca_importUpdates('http://stabc:8001',
'/tmp/all-HelloWorld_rev1.0.jar', 'HelloWorld', '1.0', partition='myPartition')
```

## sca_exportSharedData

Command Category: Application Export and Import Commands

Use with WLST: Offline

### Description

Exports shared data of a given pattern into a JAR file.

### Syntax

```
sca_exportSharedData(serverURL, jarFile, pattern, [user], [password])
```

| Argument | Definition |
|---|---|
| *serverURL* | URL of the server that hosts the SOA Infrastructure application (for example, `http://stabc:8001`). |
| *jarFile* | Absolute path of a JAR file to generate. |
| *pattern* | The file pattern supported by MDS transfer APIs. Use the semicolon delimiter (`;`) if more than one pattern is specified. Exclude the shared data namespace `/apps` in the pattern. For example:<br>`/Project1/**;/Project2/**`<br>This example exports all documents under `/apps/Project1` and `/apps/Project2`. |

| Argument | Definition |
|---|---|
| *user* | Optional. The user name for accessing the server when basic configuration is configured. Use the following syntax for this argument:<br>`user='username'` |
| *password* | Optional. The password for accessing the server when basic configuration is configured. Use the following syntax for this argument:<br>`password='password'` |

### Examples

The following example exports shared data of a given pattern into a JAR file.

```
wls:/offline/mydomain/ServerConfig> sca_exportSharedData('http://stabc:8001',
'/tmp/MySharedData.jar', '/Project1/**')
```

## sca_removeSharedData

Command Category: Application Export and Import Commands

Use with WLST: Offline

### Description

Removes a top-level shared data folder, even if there are composites deployed in the service engine.

### Syntax

```
sca_removeSharedData(serverURL, folderName, [user], [password])
```

| Argument | Definition |
|---|---|
| *serverURL* | URL of the server that hosts the SOA Infrastructure application (for example, `http://stabc:8001`). |
| *folderName* | The name of a top-level shared data folder to be removed. |
| *user* | Optional. The user name for accessing the server when basic configuration is configured. Use the following syntax for this argument:<br>`user='username'` |
| *password* | Optional. The password for accessing the server when basic configuration is configured. Use the following syntax for this argument:<br>`password='password'` |

### Examples

The following example removes the top-level shared data `Project1` folder.

```
sca_removeSharedData('http://stabc:8001', 'Project1')
```

# SOA Composite Application MBean-Based Export and Import Commands

Use the deployment commands, listed in Table 2-10, to export and import SOA composite applications on the server-based composite store MBean (`CompositeStoreMXBean`).

***Table 2-10 SOA Composite Application Export and Import Commands for WLST Configuration***

| Use this command... | To... | Use with WLST... |
|---|---|---|
| sca_exportCompositeMb | Export a SOA composite application into a SAR file. | Online |
| sca_exportUpdatesMb | Export postdeployment changes of a SOA composite application into a JAR file. | Online |
| sca_importUpdatesMb | Import postdeployment changes of a SOA composite application. | Online |
| sca_exportSharedDataMb | Export shared data of a given pattern into a JAR file. | Online |

If you use this option, note that the file generated in the export commands and the file read in the import command must be on the host where the server is running (either an Oracle WebLogic Administration Server or a managed SOA server).

The composite store MBean is registered as both a server runtime MBean of the SOA server and as a domain runtime MBean of the Oracle WebLogic Administration Server, which allows the import and export to continue working while SOA servers are down. Only WLST commands are provided for using the composite store MBean; there are no `ant` commands.

You must run the `connect()` command to connect to either a SOA server or an Oracle WebLogic Administration Server.

```
wls:offline>connect('weblogic', 'password', 't3://stabc:8001')
```

If you use the domain runtime MBean while the SOA servers are down, you must run the `domainRuntime()` command.

```
wls:offline>connect('weblogic', 'password', 't3://stabc:7001')
wls:/soainfra/serverConfig>domainRuntime()
```

## sca_exportCompositeMb

Command Category: Application Export and Import Commands

Use with WLST: Online

### Description

Exports a SOA composite application into a SAR file.

### Syntax

```
sca_exportCompositeMb(updateType, sarFile, compositeName, revision)
```

| Argument | Definition |
|---|---|
| *updateType* | Type of postdeployment changes to be exported:<br>• `all`: All postdeployment changes are included.<br>• `property`: Property changes are included (binding component properties, composite properties such as audit level settings and payload validation status, and policy attachments).<br>• `runtime`: Postdeployment runtime changes are included (rules dictionary and domain value maps (DVMs)). |
| *sarFile* | Absolute path of a SAR file to generate. |
| *compositeName* | Name of the composite to export. |
| *revision* | Revision of the composite to export. |

### Examples

This example exports composite without including any postdeployment changes.

```
wls:/mydomain/ServerConfig> sca_exportCompositeMb('none', '/tmp/sca_HelloWorld_
rev1.0.jar', 'HelloWorld', '1.0')
```

This example exports a composite with all postdeployment updates.

```
wls:/mydomain/ServerConfig> sca_exportCompositeMb('all', '/tmp/sca_HelloWorld_
rev1.0-all.jar', 'HelloWorld','1.0')
```

This example exports a composite with property postdeployment updates.

```
wls:/mydomain/ServerConfig> sca_exportCompositeMb('property', '/tmp/sca_
HelloWorld_rev1.0-prop.jar', HelloWorld', '1.0')
```

This example exports a composite with runtime/metadata postdeployment updates.

```
wls:/mydomain/ServerConfig> sca_exportCompositeMb('runtime', '/tmp/sca_HelloWorld_
rev1.0-runtime.jar','HelloWorld', '1.0')
```

## sca_exportUpdatesMb

Command Category: Application Export and Import Commands

Use with WLST: Online

### Description

Exports postdeployment changes of a SOA composite application into a JAR file.

### Syntax

```
sca_exportUpdatesMb(updateType, jarFile, compositeName, revision)
```

| Argument | Definition |
|---|---|
| *updateType* | Type of postdeployment changes to be exported: `all`, `property`, or `runtime`. |
| *jarFile* | Absolute path of a JAR file to generate. |

| Argument | Definition |
|---|---|
| *compositeName* | Name of the composite to export. |
| *revision* | Revision of the composite to export. |

### Examples

The following example exports all postdeployment updates.

```
wls:/mydomain/ServerConfig>  sca_exportUpdatesMb('all',
'/tmp/all-HelloWorld_rev1.0.jar','HelloWorld', '1.0')
```

The following example exports property postdeployment updates.

```
wls:/mydomain/ServerConfig> sca_exportUpdatesMB('property',
'/tmp/prop-HelloWorld_ rev1.0.jar', 'HelloWorld', '1.0')
```

The following example exports runtime/metadata postdeployment updates.

```
wls:/mydomain/ServerConfig> sca_exportUpdatesMB('runtime',
'/tmp/runtime-HelloWorld_ rev1.0.jar', 'HelloWorld', '1.0')
```

## sca_importUpdatesMb

Command Category: Application Export and Import Commands

Use with WLST: Online

### Description

Imports postdeployment changes of a SOA composite application.

### Syntax

```
sca_importUpdatesMb(jarFile, compositeName, revision)
```

| Argument | Definition |
|---|---|
| *jarFile* | Absolute path of a JAR file that contains postdeployment changes. |
| *compositeName* | Name of the composite to which the postdeployment changes are imported. |
| *revision* | Revision of the composite to which the postdeployment changes are imported. |

### Examples

The following example imports postdeployment changes of a SOA composite application.

```
wls:/mydomain/ServerConfig> sca_importUpdatesMb('/tmp/all-HelloWorld_rev1.0.jar',
'HelloWorld', '1.0')
```

## sca_exportSharedDataMb

Command Category: Application Export and Import Commands

Use with WLST: Online

### Description

Exports shared data of a given pattern into a JAR file.

### Syntax

```
sca_exportSharedDataMb(jarFile, pattern)
```

| Argument | Definition |
|---|---|
| *jarFile* | Absolute path of a JAR file to generate. |
| *pattern* | The file pattern supported by MDS transfer APIs. Use the semicolon delimiter (;) if more than one pattern is specified. Exclude the shared data namespace /apps in the pattern. For example:<br>/Project1/**;/Project2/**<br>This example exports all documents under /apps/Project1 and /apps/Project2. |

### Examples

This example exports shared data of given pattern into a JAR file.

```
wls:/mydomain/ServerConfig> sca_exportSharedDataMb('/tmp/MySharedData.jar',
'/Project1/**')
```

## SOA Composite Application Partition Management Commands

Use the deployment commands, listed in Table 2-11, to manage partitions. Partitioning enable you to logically group different revisions of your SOA composite applications into separate sections. This is similar to the concept of domains in the 10.1.*x* releases of Oracle BPEL Process Manager.

***Table 2-11 SOA Composite Application Partition Management Commands for WLST Configuration***

| Use this command... | To... | Use with WLST... |
|---|---|---|
| sca_createPartition | Create a partition. | Online |
| sca_deletePartition | Undeploy all SOA composite applications in a partition before deleting the partition. | Online |
| sca_startCompositesInPartition | Start all SOA composite applications in a partition. | Online |
| sca_stopCompositesInPartition | Stop all SOA composite applications in a partition. | Online |
| sca_activateCompositesInPartition | Activate all SOA composite applications in a partition. | Online |
| sca_retireCompositesInPartition | Retire all SOA composite applications in a partition. | Online |
| sca_listPartitions | List all partitions in the SOA Infrastructure. | Online |

*Table 2-11 (Cont.) SOA Composite Application Partition Management Commands for WLST Configuration*

| Use this command... | To... | Use with WLST... |
|---|---|---|
| sca_listCompositesInPartition | List all composites in a specific partition. | Online |
| sca_createPartitionWMG | Create a partition and associate it with a work manager group. | Online |

## sca_createPartition

Command Category: Application Partition Management Commands

Use with WLST: Online

### Description

Creates a partition.

### Syntax

```
sca_createPartition(partitionName)
```

| Argument | Definition |
|---|---|
| *partitionName* | The name of the partition. |

### Examples

This example creates a partition named `myPartition`.

```
wls:/mydomain/ServerConfig> sca_createPartition('myPartition')
```

## sca_deletePartition

Command Category: Application Partition Management Commands

Use with WLST: Online

### Description

Undeploys all composites in a partition before deleting the partition.

### Syntax

```
sca_deletePartition(partitionName)
```

| Argument | Definition |
|---|---|
| *partitionName* | The name of the partition. |

### Examples

This example undeploys all composites in the myPartition partition before deleting the partition.

```
wls:/mydomain/ServerConfig> sca_deletePartition('myPartition')
```

## sca_startCompositesInPartition

Command Category: Application Partition Management Commands

Use with WLST: Online

### Description

Starts all composites in a partition.

### Syntax

```
sca_startCompositesInPartition(partitionName)
```

| Argument | Definition |
|---|---|
| *partitionName* | The name of the partition. |

### Examples

This example starts all composites in the myPartition partition.

```
wls:/mydomain/ServerConfig> sca_startCompositesInPartition('myPartition')
```

## sca_stopCompositesInPartition

Command Category: Application Partition Management Commands

Use with WLST: Online

### Description

Stops all composites in a partition.

### Syntax

```
sca_stopCompositesInPartition(partitionName)
```

| Argument | Definition |
|---|---|
| *partitionName* | The name of the partition. |

### Examples

This example stops all composites in the myPartition partition.

```
wls:/mydomain/ServerConfig> sca_stopCompositesInPartition('myPartition')
```

## sca_activateCompositesInPartition

Command Category: Application Partition Management Commands

Use with WLST: Online

### Description

Activates all composites in a partition.

### Syntax

```
sca_activateCompositesInPartition(partitionName)
```

| Argument | Definition |
|---|---|
| *partitionName* | The name of the partition. |

### Examples

This example activates all composites in the myPartition partition.

```
wls:/mydomain/ServerConfig> sca_activateCompositesInPartition('myPartition')
```

## sca_retireCompositesInPartition

Command Category: Application Partition Management Commands

Use with WLST: Online

### Description

Retires all composites in a partition.

### Syntax

```
sca_retireCompositesInPartition(partitionName)
```

| Argument | Definition |
|---|---|
| *partitionName* | The name of the partition. |

### Examples

This example retires all composites in the myPartition partition.

```
wls:/mydomain/ServerConfig> sca_retireCompositesInPartition('myPartition')
```

## sca_listPartitions

Command Category: Application Partition Management Commands

Use with WLST: Online

### Description

Lists all partitions in the SOA Infrastructure.

### Syntax

```
sca_listPartitions()
```

### Examples

This example lists all partitions in the SOA Infrastructure.

```
wls:/mydomain/ServerConfig> sca_listPartitions()
```

## sca_listCompositesInPartition

Command Category: Application Partition Management Commands

Use with WLST: Online

### Description

Lists all composites in a partition.

### Syntax

```
sca_listCompositesInPartition(partitionName)
```

| Argument | Definition |
|---|---|
| *partitionName* | The name of the partition. |

### Examples

This example lists all composites in the `myPartition` partition.

```
sca_listCompositesInPartition(myPartition)
```

## sca_createPartitionWMG

Command Category: Application Partition Management Commands

Use with WLST: Online

### Description

Creates a partition and associates it with a work manager group. The work manager group is created if it does not exist.

### Syntax

```
sca_createPartitionWMG(partitionName, workmanagerGroupName,
workmanagerGroupDescription)
```

| Argument | Definition |
|---|---|
| *partitionName* | The name of the partition to be created. |
| *workmanagerGroupName* | The name of the work manager group to use. The work manager group is created if it does not exist. |
| *workmanagerGroupDescription* | The description of the work manager group if it is to be created. |

### Examples

This example creates a partition named `myPartition` and associates it with a new work manager group named `myWMG`. Since this work manager group is being created, a description is provided.

```
sca_createPartitionWMG('myPartition', 'myWMG', 'my new workmanagerGroup')
```

# SOA Composite Application Offline Deployment Management Commands

Use the deployment commands, listed in Table 2-12, to manage offline deployments of SOA composite applications and shared data in the following file:

```
$domain/config/fmwconfig/composite-offline-deployments-default.xml
```

When you restart the SOA server, the SOA composite applications and shared data registered in the file are deployed.

***Table 2-12 SOA Composite Application Offline Deployment Management Commands for WLST Configuration***

| Use this command... | To... | Use with WLST... |
|---|---|---|
| sca_registerCompositeOfflineDeployment | Register offline deployments of SOA composite applications and shared data. | Offline |
| sca_unregisterCompositeOfflineDeployment | Unregister offline deployments of SOA composite applications and shared data | Offline |

Note the following guidelines when using these WLST commands:

- If the same SOA composite application is registered again and the corresponding SAR file has the same name, the composite is not redeployed. The composite is bypassed during offline deployment. This is because offline deployment does not support composite redeployments.
- Ensure that your SOA composite application JAR name is prefixed with `sca_`. The `sca_registerCompositeOfflineDeployment` command registers either shared data deployment or composite deployment. The composite SAR file is *always* named as `sca_*.jar`. Only JAR files that start with `sca_` are considered composite SAR files. JAR files that do not start with `sca_` are treated as a shared data file.
- When you register a newer revision of a SOA composite application (for example, revision 2.0 of `sca_HelloWorld.jar`) with the `sca_registerCompositeOfflineDeployment` command and then restart the server, the previous default SOA composite application revision (for example, revision 1.0 of `sca_HelloWorld.jar`) is retired. Any runtime changes that were performed in the older 1.0 revision with a runtime tool such as Oracle SOA Composer are merged into the newer 2.0 revision.

For information about offline deployment with configuration files, see Section "Deploying SOA Composite Applications with No Servers Running" of *Developing SOA Applications with Oracle SOA Suite*.

For more information about SAR file naming conventions, see Section "Deployed Service Archives" of *Developing SOA Applications with Oracle SOA Suite*.

## sca_registerCompositeOfflineDeployment

Command Category: Application Offline Deployment Management Commands

Use with WLST: Offline

### Description

Registers offline deployments of SOA composite applications and shared data in the `composite-offline-deployments-default.xml` file. Registration enables the SOA composite applications and shared data listed in the file to be deployed when the SOA server is restarted.

### Syntax

```
sca_registerCompositeOfflineDeployment(domainDir, fileLocation, partition)
```

| Argument | Definition |
|---|---|
| *domainDir* | The absolute path to the server domain. |
| *fileLocation* | The absolute path of a composite SAR file or shared data JAR file. |
| *default* | The partition in which to deploy the composite, This argument is optional. If not specified, the default partition named `default` is used. |

### Examples

This example specifies the domain directory and the file location for the `sca_HelloWorld.jar` composite.

```
wls:/mydomain/ServerConfig> sca_registerCompositeOfflineDeployment
('/scratch/aime1/fmwhome12/user_projects/domains/soainfra',
'/tmp/sca_HelloWorld.jar')
```

This example specifies the domain directory, the file location for the `sca_App1.jar` composite, and the partition `myPartition`.

```
wls:/mydomain/ServerConfig> sca_registerCompositeOfflineDeployment
('/scratch/aime1/fmwhome12/user_projects/domains/soainfra','/tmp/sca_App1.jar',
partition='myPartition')
```

This example specifies the domain directory and the file location for the `shareddata.jar` shared data file.

```
wls:/mydomain/ServerConfig> sca_registerCompositeOfflineDeployment
('/scratch/aime1/fmwhome12/user_projects/domains/soainfra',
'/tmp/shareddata.jar')
```

## sca_unregisterCompositeOfflineDeployment

Command Category: Application Offline Deployment Management Commands

Use with WLST: Offline

### Description

Unregisters (removes) the SOA composite application or shared data from the `$domain/config/fmwconfig/composite-offline-deployments-default.xml` file so that it is not processed during a server restart. The composite or shared data is not checked during the offline deployment process. This does not impact the existing SOA composite applications. It does not undeploy or retire the composite when the server restarts.

### Syntax

```
sca_unregisterCompositeOfflineDeployment(domainDir, fileLocation)
```

| Argument | Definition |
|---|---|
| *domainDir* | The absolute path to the server domain. |
| *fileLocation* | The absolute path of a composite SAR file or a shared data JAR file. |

### Examples

This example specifies the domain directory and the `sca_HelloWorld.jar` SAR file.

```
wls:/mydomain/ServerConfig> sca_unregisterCompositeOfflineDeployment
('/scratch/aime1/fmwhome12/user_projects/domains/soainfra','/tmp/sca_HelloWorld.jar')
```

This example specifies the domain directory and the `shareddata.jar` shared data file.

```
wls:/mydomain/ServerConfig> sca_unregisterCompositeOfflineDeployment('/scratch/aime1/
fmwhome12/
user_projects/domains/soainfra','/tmp/shareddata.jar')
```

# 3

# Oracle Business Process Management Custom WLST Commands

This chapter lists and describes the custom WLST commands for Oracle Business Process Management.

## BPMLifecycleAdmin Command Group

Table 3-1 lists and describes the BPMLifecycleAdmin commands for project lifecycle administration.

***Table 3-1 BPMLifecycleAdmin Commands for Project Lifecycle Administration***

| Use this command... | To... | Use with WLST... |
|---|---|---|
| create_public_share | Create a public share | Offline |
| unlock_public_share | Unlock a public share | Offline |
| export_public_share | Export a public share to the file system | Offline |
| delete_public_share | Delete a public share | Offline |
| publish_template | Publish a template to MDS | Offline |
| export_template | Export a template to the file system | Offline |
| delete_template | Delete a template from MDS | Offline |

### create_public_share

Command Category: BPMLifecycleAdmin Commands

Use with WLST: Offline

#### Description

Use this command to create a public share from a template. The template must exist in MDS.

#### Syntax

```
create_public_share(composerUser, composerPassword, connectionURL, templateName,
publicshareId, mdsconfigLocation, [Override], [oracleHome] )
```

| Argument | Definition |
|---|---|
| *composerUser* | The Business Process Composer user who performs the current operation. |
| *composerPassword* | BPM Composer user's password |
| *connectionURL* | JNDI connection URL to the security server service in format *host*:*port* |
| *templateName* | Name of the template in MDS |
| *publicshareId* | Name of the public share to be created |
| *mdsconfigLocation* | Location of the `mds-config.xml` to be used to connect to MDS |
| `projectLocation` | The path where the public share will be created. If the path does not exists it will be created. The root is '/'. |
| *Override* | Enables you to override the public share if a public share exists in MDS with the same name. The template is not overwritten when you execute this command. |
| *oracleHome* | Optional. The Oracle home to be used. |

### Examples

The following example creates a public share named Sample_PublicShare. It is based on the template with name `Sample_Template`. The name of the public share is `Sample_PublicShare`, and the location of the `mds-config.xml` file is `/tmp/mds-config.xml`.

```
create_public_share('user_name', 'password', 'host:port','Sample_Template',
'Sample_PublicShare','/tmp/mds-config.xml')
```

The following example creates a public share named `Sample_PublicShare`. It is based on the template named `Sample_Template` that exists in MDS. The public share, not the template, is overridden. The location of the `mds-config.xml` file is `/tmp/mds-config.xml`.

```
create_public_share('user_name', 'password', 'host:port','Sample_Template',
'Sample_PublicShare','/tmp/mds-config.xml','true')"
```

## unlock_public_share

Command Category: BPMLifecycleAdmin Commands

Use with WLST: Offline

### Description

Use this command to unlock a public share. For example, when you create project by using the Ant task `create_public_share` command, the project is created as locked. You can then unlock it by using the `unlock_public_share` command.

A lock is also set by enabling or disabling the check box **enable sharing** in the project creation page in Oracle Business Process Composer.

It is also released when the user publishes a project from Business Process Composer.

The public share must exist in MDS.

### Syntax

```
unlock_public_share(composerUser, composerPassword, connectionURL, publicshareId,
mdsconfigLocation, [oracleHome] )
```

| Argument | Definition |
|---|---|
| *composerUser* | The Business Process Composer user who performs the current operation. |
| *composerPassword* | BPM Composer user's password |
| *connectionURL* | JNDI connection URL to the security server service in format *host*:*port* |
| *publicshareId* | Name of the public share to be unlocked |
| *mdsconfigLocation* | Location of the `mds-config.xml` to be used to connect to MDS |
| *oracleHome* | Optional. The Oracle home to be used |

### Example

The following example unlocks a public share named `Sample_PublicShare`. The location of the `mds-config.xml` file is `/tmp/mds-config.xml`.

```
unlock_public_share('user_name', 'password', 'host:port','Sample_PublicShare', '/tmp/
mds-config.xml')
```

## export_public_share

Command Category: BPMLifecycleAdmin Commands

Use with WLST: Offline

### Description

Use this command to export the public share from MDS to the file system.

### Syntax

```
export_public_share(composerUser, composerPassword, connectionURL,
publicshareId,fsLocation, mdsconfigLocation, [oracleHome] )
```

| Argument | Definition |
|---|---|
| *composerUser* | The Business Process Composer user who performs the current operation. |
| *composerPassword* | BPM Composer user's password |
| *connectionURL* | JNDI connection URL to the security server service in format *host*:*port* |
| *publicshareId* | Name of the public share to be exported |
| *fsLocation* | File system location where the project is to be downloaded |
| *mdsconfigLocation* | Location of the `mds-config.xml` to be used to connect to MDS |
| *oracleHome* | Optional. The Oracle home to be used |

### Example

The following example specifies the public share name as Sample_PublicShare, the file system location as /tmp, and the location of the `mds-config.xml` file as `/tmp/mds-config.xml`.

```
export_public_share('user_name', 'password', 'host:port','Sample_PublicShare','/
tmp', '/tmp/mds-config.xml')
```

## delete_public_share

Command Category: BPMLifecycleAdmin Commands

Use with WLST: Offline

### Description

Use this command to delete a public share from MDS. Executing this command requires that the public share is not locked.

### Syntax

```
delete_public_share(composerUser, composerPassword, connectionURL, publicshareId,
mdsconfigLocation, [releaseLock], [oracleHome] )
```

| Argument | Definition |
|---|---|
| *composerUser* | The Business Process Composer user who performs the current operation. |
| *composerPassword* | BPM Composer user's password |
| *connectionURL* | JNDI connection URL to the security server service in format *host*:*port* |
| *publicshareId* | Name of the public share to be deleted |
| *mdsconfigLocation* | Location of the `mds-config.xml` to be used to connect to MDS |
| *releaseLock* | Optional. If the public share is locked, this lock can be released and the delete operation completed. You can set this attribute to either `true` or `false`. If not specified, default value is `false`. |
| *oracleHome* | Optional. The Oracle home to be used |

### Examples

The following example specifies the name and location of a public share to be deleted.

```
delete_public_share('Sample_PublicShare','/tmp/mds-config.xml')
```

The following example specifies the name and location of a public share to be deleted, and that the public share should be deleted even if locked.

```
delete_public_share('user_name', 'password', 'host:port','Sample_PublicShare','/tmp/
mds-config.xml','true')
```

## publish_template

Command Category: BPMLifecycleAdmin Commands

Use with WLST: Offline

### Description

Use this command to publish the template from the file system to MDS.

### Syntax

```
publish_template(composerUser, composerPassword, connectionURL,
templateName,fsLocation, mdsconfigLocation, [Override], [oracleHome] )
```

| Argument | Definition |
|---|---|
| *composerUser* | The Business Process Composer user who performs the current operation. |

| Argument | Definition |
|---|---|
| *composerPassword* | BPM Composer user's password |
| *connectionURL* | JNDI connection URL to the security server service in format *host*:*port* |
| *templateName* | Name of the template to be published |
| *fsLocation* | File system location of the template project |
| *mdsconfigLocation* | Location of the `mds-config.xml` to be used to connect to MDS |
| `projectLocation` | The path where the public share will be created. If the path does not exists it will be created. The root is '/' . |
| *Override* | When you publish a template in MDS, this attribute enables you to override an existing template with the same name. Can either be 'true' or 'false'. If not specified, default value is 'false'. |
| *oracleHome* | Optional. The Oracle home to be used |

### Example

The following example publishes a template named `Sample_Template_Name_MDS`. to the root folder.

```
f('user_name', 'password', 'host:port','Sample_Template','/tmp/MyTemplate','/',
'/tmp/mds-config.xml')
```

The following example publishes a template named `Sample_Template_Name_MDS`.to the `'/WorkingOn/'` folder.

```
publish_template('user_name', 'password', 'host:port','Sample_Template','/tmp/
MyTemplate','/WorkingOn', '/tmp/mds-config.xml')
```

## export_template

Command Category: BPMLifecycleAdmin Commands

Use with WLST: Offline

### Description

Use this command to export the template from MDS to the file system.

### Syntax

```
export_template(composerUser, composerPassword, connectionURL, templateName,
fsLocation, mdsconfigLocation, [oracleHome] )
```

| Argument | Definition |
|---|---|
| *composerUser* | The Business Process Composer user who performs the current operation. |
| *composerPassword* | BPM Composer user's password |
| *connectionURL* | JNDI connection URL to the security server service in format *host*:*port* |
| *templateName* | Name of the template to be exported |
| *fsLocation* | File system location where the project is to be downloaded |
| *mdsconfigLocation* | Location of the `mds-config.xml` to be used to connect to MDS |
| *oracleHome* | Optional. The Oracle home to be used |

### Example

The following example specifies the template name as `Sample_Template`, the file system location as `/tmp`, and the location of the `mds-config.xml` file as `/tmp/mds-config.xml`.

```
export_template('user_name', 'password', 'host:port','Sample_Template','/tmp','/tmp/
mds-config.xml')
```

## delete_template

Command Category: BPMLifecycleAdmin Commands

Use with WLST: Offline

### Description

Use this command to delete the template from MDS.

### Syntax

```
delete_template(composerUser, composerPassword, connectionURL,
templateName,mdsconfigLocation, [oracleHome] )
```

| Argument | Definition |
|---|---|
| *composerUser* | The Business Process Composer user who performs the current operation. |
| *composerPassword* | BPM Composer user's password |

| Argument | Definition |
|---|---|
| *connectionURL* | JNDI connection URL to the security server service in format *host*:*port* |
| *templateName* | Name of the template to be deleted |
| *fsLocation* | File system location of the template project |
| *mdsconfigLocation* | Location of the `mds-config.xml` to be used to connect to MDS |
| `projectLocation` | The path where the public share will be created. If the path does not exists it will be created. The root is '/' . |
| *oracleHome* | Optional. The Oracle home to be used |

## Example

The following example deletes the template named `Sample_template` from MDS.

```
delete_template('weblogic', 'welcome1', 'host:port','/Sample_template','/tmp/mds-config.xml')
```

# 4

# Oracle Enterprise Scheduler Custom WLST Commands

Use the Oracle Enterprise Scheduler (ESS) commands to manage Oracle Enterprise Scheduler jobs. Many of the commands have analogous convenience shell scripts to make execution simpler. The native commands are described in Oracle Enterprise Scheduler Custom Native Commands and the shell scripts are described in Oracle Enterprise Scheduler Convenience Scripts.

**Note:**

To use these Oracle Enterprise Scheduler custom WLST commands, you must invoke the WLST script from the Oracle Common home. See "Using Custom WLST Commands" in the *Administering Oracle Fusion Middleware.*

This chapter includes the following sections:



## Native Invocation

The following steps describe how to run the native WLST commands described in this chapter. Convenience shell scripts are available for many of these commands and are described in Oracle Enterprise Scheduler Convenience Scripts.

1. Change directory to:

   `$ORACLE_HOME/common/bin`

2. Place the following JAR archives are in your class path:

   - *MW_HOME*`/oracle_common/modules/oracle.jmx_11.1.1/jmxframework.jar`
   - *MW_HOME*`/WLS_HOME/lib/weblogic.jar`
   - *MW_HOME*/*ORACLE_HOME*`/ess/lib/ess-admin.jar`

3. Execute one of the following commands:
   - `wlst.sh` (Linux)
   - `wlst.cmd` (Windows)
4. Connect to the WLS Admin Server. For example:

```
connect('weblogic', 'weblogic1', 't3://hostabc:7001')
```

5. Invoke the WLST commands.

## Enabling and Disabling Print Statements

You can explicitly enable or disable Oracle Enterprise Scheduler print output as shown below. For interactive use, you must enable printing. By default, Oracle Enterprise Scheduler printing is enabled.

- `enableESSPrint()` — Enables print output for Oracle Enterprise Scheduler WLST commands.
- `disableESSPrint()` — Disables print output for Oracle Enterprise Scheduler WLST commands.

## Getting Help

While in the WLST console, you can get the list of available Oracle Enterprise Scheduler native commands by typing:

```
wls:/offline> help('OracleScheduler')
```

For usage information about any command, type `help('command_name')`. For example:

```
wls:/offline> help('manageSchedulerRuntimeConfig')
```

# Oracle Enterprise Scheduler Custom Native Commands

Use the Oracle Enterprise Scheduler commands listed in Table 4-1 to manage the Oracle Enterprise Scheduler server, configuration, job requests, and logs. In the third column, "online" means the command can only be used when connected to a running administration server. "Offline" means the command can only be used when not connected to a running server.

***Table 4-1 Oracle Enterprise Scheduling Service Basic Commands***

| Use this command... | To... | Use with WLST... |
|---|---|---|
| `manageSchedulerRuntimeConfig` | Add, modify, delete and display various configuration parameters. | Online |
| `getSchedulerRequestContent` | Get the log and output data files for a request after its execution is completed. | Online |
| `querySchedulerRequests` | Search and list requests based upon hosting application name, state or elapsed time of execution. | Online |

*Table 4-1 (Cont.) Oracle Enterprise Scheduling Service Basic Commands*

| Use this command... | To... | Use with WLST... |
|---|---|---|
| `manageSchedulerRequest` | Manually cancel, recover, or complete request state. | Online |
| `manageSchedulerServer` | Start, stop or get status of the Oracle Enterprise Scheduler application running on the server. | Online |
| `submitSchedulerRequest` | Submit a job request to Oracle Enterprise Scheduler for execution. | Online |
| `manageSchedulerJobDefn` | Manage (show, create, delete, customize, update) an Oracle Enterprise Scheduler job definition. | Online |
| `manageSchedulerSchedule` | Manage (show, create, delete, update) a schedule definition in Oracle Enterprise Scheduler. | Online |

## Manage (Add/Delete/Modify/Get) Configuration Parameters

Command Category: Application Management Commands

Use with WLST: Online

### Description

Adds, modifies, deletes and displays various configuration parameters.

### Syntax

```
manageSchedulerRuntimeConfig(app,type,[operation],[name],[val])
```

| Argument | Definition |
|---|---|
| app | The hosting application name for managing run time configuration. |
| *type* | The property type. Can be either APP or ESS. |
| operation | Optional. The operation to perform. One of the following:<br>• Add (`add`)<br>• Modify (`mod`)<br>• Delete (`del`)<br>• Get (`get`)<br>• Get All (`getall`) |
| name | (Optional for the `getall` operation) The name of the configuration parameter. |
| *val* | (Optional for the `del/get/getall` operations) Value to set for the parameter. |

### Examples

- Add an ENV parameter `foo` with value "bar":

```
manageSchedulerRuntimeConfig("myapp", "APP", operation="add", name="foo",
val="bar")
```

- Get the value of ENV parameter `foo`:

```
manageSchedulerRuntimeConfig("myapp", "APP", operation="get", name="foo")
```

- Get the list of all ENV parameters:

```
manageSchedulerRuntimeConfig("myapp", "APP", operation="getall")
manageSchedulerRuntimeConfig("myapp", "APP")
```

- Change the value of the ENV parameter `foo` to "barone":

```
manageSchedulerRuntimeConfig("myapp", "APP", operation="mod", name="foo",
val="barone")
```

- Delete the ENV parameter `foo`:

```
manageSchedulerRuntimeConfig("myapp", "APP", operation="del", name="foo")
```

- Show all parameters of type ESS:

```
manageSchedulerRuntimeConfig("myapp", "ESS")
```

### Related Shell Script

essManageRuntimeConfig

## Get Log and Output Content of a Request

Command Category: Application Management Commands

Use with WLST: Online

### Description

Gets the log and output data files for a request after its execution is completed.

### Syntax

```
getSchedulerRequestContent(requestId,contentType,[logLines],
[outDir])
```

| Argument | Definition |
|---|---|
| *requestId* | request ID |
| *contentType* | Type of content to handle. Can be one of the following;<br>• LOG<br>• OUTPUT<br>• BINARY_OUTPUT<br>• TEXT_OUTPUT<br>By default, the command checks for both BINARY_OUTPUT and TEXT_OUTPUT. |

| Argument | Definition |
|---|---|
| *logLines* | Optional. The number of lines to be read from the request log. The default value is `1000`. |
| *outDir* | Optional. The absolute path of the directory in which to dump the output files. By default, the command uses the current directory. |

### Examples

- Get the request log for request ID 123:

```
getSchedulerRequestContent(123, "LOG")
```

- Get all the output of request 123:

```
getSchedulerRequestContent(123, "OUTPUT")
```

- Get all the output of request 123 and save it in directory `/tmp`:

```
getSchedulerRequestContent(123, "OUTPUT", outDir="/tmp")
```

- Get all the text output of request 123 and save it in directory `/tmp`:

```
getSchedulerRequestContent(123, "TEXT_OUTPUT", outDir="/tmp")
```

- Get all the binary output of request 123 and save it in directory `/tmp`:

```
getSchedulerRequestContent(123, "BINARY_OUTPUT", outDir="/tmp")
```

- Get first 100 lines of the request log for request id 123:

```
getSchedulerRequestContent(123, "LOG", logLines=100)
```

### Related Shell Script

essGetOutputContent

## Search and List Requests

Command Category: Application Management Commands

Use with WLST: Online

### Description

Searches and lists requests based upon hosting application name, state or elapsed time of execution. This command can be used to find long running requests.

### Syntax

```
querySchedulerRequests([app],[state],[days],[hours],[minutes])
```

| Argument | Definition |
|---|---|
| *app* | Optional. The name of the hosting application |

| Argument | Definition |
|---|---|
| *state* | Optional. Specifies the request state. Can be one of the following:<br>• WAIT<br>• READY<br>• RUNNING<br>• COMPLETED<br>• BLOCKED<br>• HOLD<br>• CANCELLING<br>• EXPIRED<br>• CANCELLED<br>• ERROR<br>• WARNING<br>• SUCCEEDED<br>• PAUSED<br>• PENDING_VALIDATION<br>• VALIDATION_FAILED<br>• SCHEDULE_ENDED<br>• FINISHED<br>• ERROR_AUTO_RETRY<br>• ERROR_MANUAL_RECOVERY<br>By default, the command specifies RUNNING. |
| *days* | Optional. Specifies time in days |
| *hours* | Optional. Specifies time in hours |
| *minutes* | Optional. Specifies time in hours |

### Examples

- Get all the requests that are in the RUNNING state:

```
querySchedulerRequests()
or
querySchedulerRequests(state="RUNNING")
```

- Get all cancelled requests:

```
querySchedulerRequests(state="CANCELLED")
```

- Get all requests running for more than two days:

```
querySchedulerRequests(days=2)
```

- Get all requests running for more than 10 hours:

```
querySchedulerRequests(hours=10)
```

- Get all requests running for the application myapp:

```
querySchedulerRequests(appName="myapp")
```

### Shell Script

essQueryRequests

## Manage Requests

Command Category: Application Management Commands

Use with WLST: Online

### Description

Manually cancel, recover, complete, or get details about requests.

### Syntax

```
manageSchedulerRequest(requestId,operation,[asyncStatus],
[statusMessage])
```

| Argument | Definition |
|---|---|
| *requestid* | Specifies the request ID |
| *operation* | Specifies which one of the following operations to perform:<br>• CANCEL<br>• RECOVER<br>• COMPLETE<br>• GETDETAIL |
| *asyncStatus* | Required for the COMPLETE operation. Specifies the status to set for the given request. Must be one of the following:<br>• SUCCESS<br>• PAUSE<br>• WARNING<br>• ERROR<br>• CANCEL<br>• UPDATE<br>• BIZ_ERROR<br>• ERROR_MANUAL_RECOVERY |
| *statusMessage* | Required for the COMPLETE operation. Specifies the qualifying status message to describe the operation |

### Examples

- Cancel request 123:

```
manageSchedulerRequest(123, "CANCEL")
```

- Recover request 123:

```
manageSchedulerRequest(123, "RECOVER")
```

- Complete request 123:

```
manageSchedulerRequest(123, "COMPLETE", asyncStatus="ERROR",
statusMessage="Completed by Admin")
```

### Related Shell Script

essManageRequests

## Manage Oracle Enterprise Scheduler Servers

Command Category: Application Management Commands

Use with WLST: Online

### Description

Starts, stops or gets the status of the Oracle Enterprise Scheduler application running on the server. Starting the Oracle Enterprise Scheduler application starts the Oracle Enterprise Scheduler processor thread so that request processing can start. Stopping Oracle Enterprise Scheduler stops or quiesces the Oracle Enterprise Scheduler processor so that no new requests are processed.

### Syntax

```
manageSchedulerServer(operation)
```

| Argument | Definition |
|---|---|
| *operation* | Specifies which one of the following operations to perform:<br>• START<br>• STOP<br>• STATUS |

### Examples

- Stop Oracle Enterprise Scheduler:

  ```
  manageSchedulerServer("STOP")
  ```

- Get the current state of the Oracle Enterprise Scheduler processor:

  ```
  manageSchedulerServer("STATUS")
  ```

### Related Shell Script

essManageServer

## Submit Job Requests to Oracle Enterprise Scheduler

Command Category: Application Management Commands

Use with WLST: Online

### Description

Submit a job request to Oracle Enterprise Scheduler for execution.

### Syntax

```
submitSchedulerRequest(appName,[jobName],[schMeth],[note],
[schName],[RschName],[schDesc],[o_time],[begin_time],
[frequency],[freqNum],[count],[end_time],[month],[week],[day],
[date],[reqParams])
```

**Note:**

While a job definition must be predefined, the schedule can be defined in an ad-hoc manner using this command.

| Argument | Definition |
|---|---|
| *appName* | Name of the hosting application for the job. |
| *jobName* | Name of the job definition to be used for current request. The path name (including the package name) must be fully qualified. If it is not fully qualified, the command uses the default package: `/oracle/apps/ess/custom`. |
| *schMeth* | Optional. Specifies which one of the following scheduling methods to use:<br>• `IMMEDIATE` (`I`) - No other parameter is required. This is the default method and is used if the *schMeth* argument is not specified.<br>• `ONCE` (`O`) - If this method is used, the *o_time* argument is required.<br>• `SCHEDULE` (`S`) - If this method is used, the *schName* argument is required.<br>• `RECURRING` (`R`) - If this method is used, the *RschName* and *frequency* arguments are required.<br>Note that the following arguments are only valid with the Recurring(`R`) method: *schDesc*, *begin_time*, *freqNum*, *count*, *end_time*, *month*, *week*, *day*, *date* and *frequency*.Parameter combinations based on 'frequency' value are:<br>If you use the recurring (`R`) method with any of the following options: SECOND \| MINUTE \| HOUR \| DAY \| WEEK, you must use the following arguments: *freqNum*, *begin_time*, [*count* \| end_time].<br>If you use the recurring (`R`) method with the MONTH option, you must use the following arguments [{*week day*} \| *date*] *begin_time* [*count* \| *end_time*].<br>If you use the recurring (`R`) method with the YEAR option, you must use the following arguments: [*month* ({*week day*} \| *date*)] *begin_time* [*count* \| *end_time*].<br>Note that when you use the recurring (`R`) method, the schedule is stored in the MDS for future use and the storage location is based on the schedule name provided.<br>See Examples for examples. |
| *note* | Optional. Specifies a short description for the job request. |
| *schName* | Optional. Specifies the name of the predefined schedule definition. The path name (including the package name) must be fully qualified. If it is not fully qualified, the command uses the default package: `/oracle/apps/ess/custom`. |
| *RschName* | Optional. Specifies the name of the schedule if you use the recurring (`R`) option to the *schMeth* argument to define the schedule. |
| *schDesc* | Optional. Specifies a string value that describes the ad-hoc schedule. By default, the value is "Custom Ad-Hoc Schedule". |

| Argument | Definition |
|---|---|
| *o_time* | Optional. Specifies the time for a "one time" execution. The format is: *HH*:*MM*:*SS*:*DD*:*MM*:*YYYY* |
| *begin_time* | Optional. Specifies the start time for a recurring schedule. The format is: *HH*:*MM*:SS:*DD*:*MM*:*YYYY* |
| *frequency* | Optional. Specifies the frequency of a recurring schedule. The valid values are: `SECOND`, `MINUTE`, `HOUR`, `DAY`, `WEEK`, `MONTH`, `YEAR` |
| *freqNum* | Optional. An integer value that specifies repeat interval of a recurring schedule. The default value is "1". |
| *count* | Optional. Specifies the maximum number of repetitions for a request. |
| *end_time* | Optional. Specifies the end time for a recurring schedule. The format is: *HH*:*MM*:*SS*:*DD*:*MM*:*YYYY* |
| *month* | Optional. Specifies the month of the year if *frequency* is specified as `YEARLY`. Valid values are: `1-12`. |
| *week* | Optional. Specifies the week of a month. Valid values are: `1-6`, `LAST`. |
| *day* | Optional. Specifies the day of the week. Valid values are: `1-7`, where "`1`" represents Monday. |
| *date* | Optional. Specifies a specific day of month. Valid values are: `1-31`, `LAST` |
| *reqParams* | Optional. Specifies other request parameters/properties specified as a dictionary. |

### Examples

The following variables are used in these examples:

| Variable | Value |
|---|---|
| `HOSTING_APP` | `EssDemoApp` |
| `SCHEDULE_NAME` | `/oracle/apps/ess/custom/MySch` |
| `JOB_DEF_NAME` | `/oracle/apps/ess/demopackage/BasicJavaJob` |

- Submit a request for immediate job execution (one-time only):

```
submitSchedulerRequest(HOSTING_APP, JOB_DEF_NAME)
```

- Submit a request for one-time job execution at a specified time:

```
submitSchedulerRequest(HOSTING_APP, JOB_DEF_NAME, schMeth='O',
o_time='21:10:30:03:05:2012')
```

- Submit a job request with a recurring schedule. The recurrence is: Execution recurs every 2 minutes, starting at time 21:10:30 on 03 May 2012, until 3 iterations complete. The additional request parameters are: *reqName* =`test` and *PRIORITY*= `10`. This recurrence also persists in the MDS repository for future use.

```
submitSchedulerRequest(HOSTING_APP, JOB_DEF_NAME, schMeth='R',
RschName=SCHEDULE_NAME, frequency='MINUTE', freqNum=2, count=3,
begin_time='21:10:30:03:05:2012', reqParams={'reqName':'test','PRIORITY':'10'})
```

- Submit a job request using a predefined schedule

```
submitSchedulerRequest(HOSTING_APP, JOB_DEF_NAME, schMeth='S',
schName=SCHEDULE_NAME)
```

- Submit a job request with a monthly recurring schedule. The description for the request is "Request with monthly schedule". The recurring schedule name and description are specified using *RschName* and *schDesc* respectively.

```
submitSchedulerRequest(HOSTING_APP, JOB_DEF_NAME, note='Request with monthly
schedule', schMeth='R', RschName=SCHEDULE_NAME+'1', schDesc='Monthly_schedule',
frequency='MONTH', freqNum=2, begin_time='21:10:30:03:05:2012', week=5, day=4,
end_time='21:10:30:04:05:2013')
```

- Submit a job request with a yearly recurring schedule. The description for the request is "Request with yearly schedule". The recurring schedule name and description are provided by *RschName* and *schDesc* respectively.

  Execution recurs every three years, on the last day of May (31st), starting at 21:10:30 on 03 May 2012, and iterates five times.

```
submitSchedulerRequest(HOSTING_APP, JOB_DEF_NAME,note='Request with yearly
schedule',schMeth='R',RschName=SCHEDULE_NAME+'2', schDesc='Yearly_schedule',
frequency='YEAR',freqNum=3,begin_time='21:10:30:03:05:2012',month=5,
date='LAST',count=5)
```

- Submit a job request with an hourly recurring schedule. This schedule is stored in the MDS. Recurrence is set to repeat every hour starting from the current time.

```
submitSchedulerRequest(HOSTING_APP, JOB_DEF_NAME,schMeth='R',
RschName=SCHEDULE_NAME,schDesc='HOURLY_schedule',frequency='HOUR')
```

### Related Shell Script

essSubmitRequest

## Manage Oracle Enterprise Scheduler Job Definitions

Command Category: Application Management Commands

Use with WLST: Online

### Description

Manage (show, create, delete, customize, update) an Oracle Enterprise Scheduler job definition.

If *operation* is set to SHOW and if *jobName* is specified, the command returns the details of this particular job definition; otherwise, the command shows a list of all job definitions that are part of the application.

If *operation* is set to CREATE, the command returns the MetadataObjectID of the newly created job definition.

If *operation* is set to DELETE, CUSTOMIZE or UPDATE, the command returns 0 or -1, depending on whether the operation succeeds or fails.

### Syntax

```
manageSchedulerJobDefn(operation,appName,[jobName],[jobType],
[desc],[props])
```

| Argument | Definition |
|---|---|
| *operation* | Specifies which of the following operations to perform on the job:<br>• SHOW (S)<br>Displays the list of job definitions that are part of a particular application. If a job name is specified, it shows the details of only that job definition.<br>If this operation is used, the *appName* argument is required and *jobName* is optional.<br>• CREATE (N)<br>Creates a new job definition in the namespace of the relevant application (supplied as a parameter) in MDS.<br>If this operation is used, the *appName*, *jobName*, and *jobType* arguments are required and *desc* and *props* are optional.<br>• DELETE (D)<br>Deletes the specified job definition from the MDS.<br>If this operation is used, the *appName* and*jobName* arguments are required.<br>• CUSTOMIZE (C)<br>Modifies the customizable properties of an existing job definition. The following customizable properties are available: SYS_retries, SYS_priority, SYS_requestCategory, SYS_request_timeout, enableTrace, enableTimeStatistics.<br>If this operation is used, the *appName*, *jobName*, and *props* arguments are required.<br>• UPDATE (U)<br>Updates an existing job definition with the specified property values (existing parameters persist if not overridden).<br>If this operation is used, the *appName*, *jobName*, and one or both of the *desc* and *props* arguments are required. |
| *appName* | Specifies the logical name of the job's hosting application. |
| *jobName* | Optional. Specifies the name of the job definition. For the create, delete, or update operations, a default package /oracle/apps/ess/custom/ is prepended to the specified name. |
| *jobType* | Optional. Specifies the name of the job type to be used to create a new job definition. If the name does not contain the package, the following default value is prefixed to the name: /oracle/as/ess/core/. |
| *desc* | Optional. Specifies a description for a new job definition. |
| *props* | Optional. Specifies other request parameters/properties specified as a dictionary. |

**Note:**

- For job definition creation, deletion and update, the value of the *jobName* argument can be a fully qualified name (that includes the package name); however, the default package /oracle/apps/ess/custom is always prefixed to it.
- As indicated by the previous note, you can only create/delete/update the job definitions present in the /oracle/apps/ess/custom namespace; prepackaged job definitions cannot be modified.
- If you do not provide a fully qualified path name for the *jobType* argument, a default package (/oracle/as/ess/core/) is prefixed to the name.

### Examples

The following variables are used in these examples:

| Variable | Value |
|---|---|
| HOSTING_APP | EssNativeHostingApp |
| JOB_DEF_PREDEF | /oracle/apps/ess/TestJob |
| JOB_DEF_NAME | TestJob_wlst |
| JOB_TYPE_NAME | JavaJobType |
| JOB_DESC | My WLST Test Defn |
| PARAMS | 'SYS_retries':1,'myParam':'xyz' |

- Show all job definitions present in the namespace of HOSTING_APP:

```
manageSchedulerJobDefn('SHOW',HOSTING_APP)
```

- Show the details of the job definition (JOB_DEF_PREDEF) part of HOSTING_APP:

```
manageSchedulerJobDefn('SHOW',HOSTING_APP,jobName=JOB_DEF_PREDEF)
```

- Create the new job definition /oracle/apps/ess/custom/TestJob_wlst in the namespace of HOSTING_APP with the *jobType* value of /oracle/as/ess/core/JavaJobType:

```
manageSchedulerJobDefn('CREATE',HOSTING_APP,jobName=JOB_DEF_NAME,
jobType=JOB_TYPE_NAME)
```

- Create a new job definition named /oracle/apps/ess/custom/TestJob_wlst in the HOSTING_APP namespace with the *jobType* /oracle/as/ess/core/JavaJobType and a description with the value of JOB_DESC:

```
manageSchedulerJobDefn('CREATE',HOSTING_APP,jobName=JOB_DEF_NAME,jobType=JOB_TYPE_
NAME,desc=JOB_DESC)
```

- Create a new job definition named /oracle/apps/ess/custom/TestJob_wlst in the HOSTING_APP namespace with the *jobType* /

`oracle/as/ess/core/JavaJobType` and a description and parameters defined by `JOB_DESC` and `PARAMS`:

```
manageSchedulerJobDefn('CREATE',HOSTING_APP,jobName=JOB_DEF_NAME,jobType=JOB_TYPE_
NAME,desc=JOB_DESC,props=PARAMS)
```

- Update the job definition `/oracle/apps/ess/custom/JOB_DEF_NAME` with the new description `JOB_DESC`:

```
manageSchedulerJobDefn('UPDATE',HOSTING_APP,jobName=JOB_DEF_NAME,desc=JOB_DESC)
```

- Update the job definition `/oracle/apps/ess/custom/JOB_DEF_NAME` with the properties `PARAMS`:

```
manageSchedulerJobDefn('UPDATE',HOSTING_APP,jobName=JOB_DEF_NAME,props=PARAMS)
```

- Delete the job definition `/oracle/apps/ess/custom/JOB_DEF_NAME` in `HOSTING_APP`:

```
manageSchedulerJobDefn('DELETE',HOSTING_APP,jobName=JOB_DEF_NAME)
```

- Customize the prepackaged job definition `JOB_DEF_PREDEF` in `HOSTING_APP` with the customizable properties SYS_retries and SYS_priority:

```
manageSchedulerJobDefn('CUSTOMIZE',HOSTING_APP,jobName=JOB_DEF_PREDEF,props={'SYS_
retries':2,'SYS_priority':3})
```

### Related Shell Script

essManageJobDefn

## Manage Oracle Enterprise Scheduler Schedule Definitions

Command Category : Application Management Commands

Use with WLST: Online

### Description

Manage (show, create, delete, update) a schedule definition in Oracle Enterprise Scheduler.

If *`operation`* is set to `SHOW` and if *`schName`* is specified, the command shows the details of this particular schedule definition; otherwise, the command shows a list of all schedule definitions that are part of the application. The command returns 0 or -1 depending on whether the operation succeeds or fails.

If *`operation`* is set to `CREATE`, the command returns the MetadataObjectID of the newly created schedule definition.

If *`operation`* is set to `DELETE` or `UPDATE`, the command returns 0 or -1, depending on whether the operation succeeds or fails.

### Syntax

```
manageSchedulerSchedule(operation,appName,[schName],[schDesc],
[begin_time],[frequency],[interval],[count],[end_time],[month],
[week],[day],[date])
```

| Argument | Definition |
|---|---|
| *operation* | Specifies which of the following operations to perform on the job:<br>• SHOW (S)<br>Displays the list of schedule definitions that are part of a particular application.<br>If this operation is used, the *appName* argument is required and *schName* is optional.<br>• CREATE (C)<br>Creates a new schedule definition in the namespace of the relevant application (supplied as a parameter) in MDS.<br>If this operation is used, the *appName*, *schName*, and *frequency* arguments are required and *schDesc* is optional.<br>• DELETE (D)<br>Deletes the specified job definition from the MDS.<br>If this operation is used, the *appName* and*schName* arguments are required.<br>• UPDATE (U)<br>Updates an existing schedule definition with the specified property values (existing parameters persist if not overridden).<br>If this operation is used, the *appName*, *schName*, and one or both of the *schDesc* and *frequency* arguments are required. |
| *appName* | Specifies the logical name of the job's hosting application. |
| *schName* | Optional. Specifies the name of the schedule definition. For the create, delete, and update operations, a default package named /oracle/apps/ess/custom/ is prepended to the name provided. |
| *schDesc* | Optional. Specifies a description for the schedule definition when creating or updating a definition. |
| *begin_time* | Optional. Specifies the start time for a recurring event. The format is: *HH*:*MM*:*SS*:*DD*:*MM*:*YYYY* |
| *frequency* | Optional. Frequency of recurrence. Valid values are: SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, YEAR<br>If you use any of the following options: SECOND, MINUTE, HOUR, DAY, WEEK, you must use the following arguments: *interval*, *begin_time*, [*count* \| *end_time*].<br>If you use the MONTH option, you must use the following arguments [(*week day*) \| *date*] [*begin_time* ] [*count* \| *end_time*].<br>If you use the YEAR option, you must use the following arguments: [*month* ({*week day*} \| *date*)] [*begin_time*] [*count* \| *end_time*]. |
| *interval* | Optional. Specifies the repeat interval used with the *frequency* option. This is an integer value with a default value of 1. |
| *count* | Optional. Specifies the maximum number of repetitions for a request. This value is an integer. |
| *end_time* | Optional. Specifies the end time for recurrence. Format: *HH*:*MM*:*SS*:*DD*:*MM*:*YYYY* |
| *month* | Optional. Specifies the month of the year if *frequency* is set to YEAR. Valid values are 1-12. |

| Argument | Definition |
|---|---|
| *week* | Optional. Specifies the week of a month. Valid values are `1-6` and `LAST`. |
| *day* | Optional. Specifies the day of a week. Valid values are `1-7`, where `1` represents Monday. |
| *date* | Optional. Specifies a day of month. Valid values are `1-31` and `LAST`. |

### Examples

The following variables are used in these examples:

| Variable | Value |
|---|---|
| `HOSTING_APP` | `EssNativeHostingApp` |
| `PREDEF_SCHEDULE` | `/oracle/apps/ess/demo/seeded/MyPredefinedSch` |
| `SCHEDULE_NAME` | `MyWlstSchedule` |
| `SCH_DESC` | `Description for WLST test schedule` |

- Show all schedule definitions present in the namespace of `HOSTING_APP`:

```
manageSchedulerSchedule('SHOW',HOSTING_APP)
```

- Show the details of the `HOSTING_APP` schedule definition `PREDEF_SCHEDULE`:

```
manageSchedulerSchedule('SHOW',HOSTING_APP,schName=PREDEF_SCHEDULE)
```

- Create a new recurring schedule definition named `/oracle/apps/ess/custom/MyWlstSchedule` in the namespace of `HOSTING_APP` and with the description `SCH_DESC`. The recurring schedule is:
  - Occur every 2 minutes
  - Start at 21:10:30 on May 03, 2012
  - Complete 3 iterations

```
manageSchedulerSchedule('CREATE',HOSTING_APP,schName=SCHEDULE_NAME,
schDesc=SCH_DESC,frequency='MINUTE',interval=2,count=3,
begin_time='21:10:30:03:05:12')
```

- Create a new recurring schedule definition named `/oracle/apps/ess/custom/MyWlstSchedule` in the namespace of `HOSTING_APP`. The recurring schedule is:
  - Occur every 2 months on Thursday of the 5th week (if applicable)
  - Start at 21:10:30 on May 03, 2012
  - Continue until 21:10:30 on May 04, 2013

```
manageSchedulerSchedule('CREATE',HOSTING_APP,schName=SCHEDULE_NAME,
frequency='MONTH',interval=2, begin_time='21:10:30:03:05:12', week=5, day=4,
end_time='21:10:30:04:05:13')
```

- Create a new recurring schedule definition named `/oracle/apps/ess/custom/MyWlstSchedule` in namespace of `HOSTING_APP`. The recurring schedule is:
  - Occur every 3 years, on the last day of May (May 31st)
  - Start at 21:10:30 on May 03, 2012
  - Complete 5 iterations

```
manageSchedulerSchedule('CREATE',HOSTING_APP,schName=SCHEDULE_NAME,
frequency='YEAR',interval=3,begin_time='21:10:30:03:05:12',month=5,date='LAST',cou
nt=5)
```

- Create a new recurring schedule definition named `/oracle/apps/ess/custom/MyWlstSchedule` in the namespace of `HOSTING_APP`. The recurring schedule is: occur hourly, starting now.

```
manageSchedulerSchedule('CREATE',HOSTING_APP,schName=SCHEDULE_NAME,
frequency='HOUR')
```

- Update the schedule definition named `/oracle/apps/ess/custom/SCHEDULE_NAME` with the description `SCH_DESC`. The recurring schedule remains unchanged.

```
manageSchedulerSchedule('UPDATE',HOSTING_APP,schName=SCHEDULE_NAME,schDesc=SCH_DES
C)
```

- Update the schedule definition named `/oracle/apps/ess/custom/SCHEDULE_NAME` with the following new recurring schedule: Occur every minute.

```
manageSchedulerSchedule('UPDATE',HOSTING_APP,schName=SCHEDULE_NAME,
frequency='MINUTE')
```

- Delete the `HOSTING_APP` schedule definition named `/oracle/apps/ess/custom/SCHEDULE_NAME`.

```
manageSchedulerSchedule('DELETE',HOSTING_APP,schName=SCHEDULE_NAME)
```

### Related Shell Script

essManageSchedule

## Oracle Enterprise Scheduler Batch Delete Requests

Job request data is kept as records in the Oracle Enterprise Scheduler runtime store tables. These job requests records are not removed from the runtime store until they are physically purged by a database administrator using SQL purge scripts. A request must be logically deleted before it can be physically purged. Oracle Enterprise Scheduler provides two ways to delete a request, either by a user performing a delete operation for a specific request, or by an Oracle Enterprise Scheduler administrator submitting a batch delete request.

The batch delete feature provides a way for an Oracle Enterprise Scheduler administrator to delete requests in a batch fashion. An Oracle Enterprise Scheduler administrator defines a set of delete criteria and submits a batch delete request using the Oracle Enterprise Scheduler runtime MBean interface. The delete criteria for the batch job are specified in the form of application request parameters. The submitted batch delete request uses a pre-defined job definition for a Java executable that is supplied by Oracle Enterprise Scheduler. The batch delete job definition metadata is defined in the ESSAPP metadata repository.

When the batch delete job runs, it determines which completed absolute parent and instance parent requests satisfy the delete criteria specified for that batch job request and deletes the request hierarchy for those requests. The batch delete job is executed in the context of ESSAPP.

## The batchDeleteSchedulerRequest Command

Command Category: Application Management Commands

Use with WLST: Online

### Description

Submits a request for a batch delete job. You can use either a pre-defined schedule (in MDS) or specify a one-time execution time. If neither of these is specified, the request starts immediately. The request parameters are used to specify the delete criteria.

When the batch delete job runs, it determines which completed absolute parent and instance parent requests satisfy the delete criteria specified for that batch job request. The batch delete job deletes the request hierarchy for those requests.

### Syntax

```
batchDeleteSchedulerRequests([desc],[schId],[start],[end],
[params])
```

| Argument | Definition |
|---|---|
| *desc* | Optional. Specifies a description for the batch delete request. |
| *schId* | Optional. Specifies the fully qualified name of the schedule (pre-defined) to use. The default package, if none is specified, is: `/oracle/as/ess/essapp/custom/`. |
| *start* | Optional. Specifies the time to start the request. Format: *HH*:*MM*:*SS*:*DD*:*MM*:*YYYY* |
| *end* | Optional. The time the request should terminate. Format: *HH*:*MM*:*SS*:*DD*:*MM*:*YYYY* |
| *params* | Optional. Request parameters specified as a dictionary. Batch Delete Parameters describes the parameters. |

#### Batch Delete Parameters

A batch delete job uses application request parameters to specify the batch delete criteria and basic result information for that delete job.The class `oracle.as.scheduler.job.BatchDeleteProperty.java` defines constants for the batch delete parameters. The delete criteria parameters are described in the following two tables.

| Parameter | Data Type | Constants |
|---|---|---|
| `CriteriaApplication` | String | `BatchDeleteProperty.CRITERIA_APPLICATION` |
| `CriteriaProduct` | String | `BatchDeleteProperty.CRITERIA_PRODUCT` |

| Parameter | Data Type | Constants |
|---|---|---|
| CriteriaJobDefn | String | BatchDeleteProperty.CRITERIA_JOBDEFN |
| CriteriaJobExecType | String | BatchDeleteProperty.CRITERIA_JOBEXECTYPE |
| CriteriaSubmitUser | String | BatchDeleteProperty.CRITERIA_SUBMIT_USER |
| CriteriaMinimumAge | Integer | BatchDeleteProperty.CRITERIA_MINIMUM_AGE |
| CriteriaRetentionAgeSuccess | Integer | BatchDeleteProperty.CRITERIA_RETENTION_AGE_SUCCESS |
| CriteriaRetentionAgeError | Integer | BatchDeleteProperty.CRITERIA_RETENTION_AGE_ERROR |
| CriteriaProcessLimit | Integer | BatchDeleteProperty.CRITERIA_RETENTION_AGE_WARNING |
| CriteriaRetentionAgeCancel | Integer | BatchDeleteProperty.CRITERIA_RETENTION_AGE_CANCEL |
| CriteriaProcessLimit | Long | BatchDeleteProperty.CRITERIA_PROCESS_LIMIT |

| Parameter | Default Value | Description |
|---|---|---|
| CriteriaApplication | null (none) | Specifies the application for which requests are to be deleted. If null, this parameter is ignored and is not used as part of the delete criteria. |
| CriteriaProduct | null (none) | Specifies the product for which requests are to be deleted. If null, this parameter is ignored and is not used as part of the delete criteria. |
| CriteriaJobDefn | null (none) | Specifies the string representation of the job definition `MetadataObjectId` for which requests are to be deleted. If null, this parameter is ignored and is not used as part of the delete criteria. |
| CriteriaJobExecType | null (none) | Specifies the string representation of the job execution type for which requests are to be deleted. If null, this parameter is ignored and is not used as part of the delete criteria. |
| CriteriaSubmitUser | null (none) | Specifies the submitting user for which requests are to be deleted. If null, this parameter is ignored and is not used as part of the delete criteria. |
| CriteriaMinimumAge | null (none) | Specifies a minimum age, in days, before a request that completed is eligible for deletion. |

| Parameter | Default Value | Description |
|---|---|---|
| `CriteriaRetentionAgeSuccess` | null (none) | Specifies a minimum age, in days, before a request that completed as success is eligible for deletion. If this parameter value is null, then the value of `CriteriaMinimumAge` is used. |
| `CriteriaRetentionAgeError` | null (none) | Specifies a minimum age, in days, before a request that completed as a failure is eligible for deletion. If this parameter value is null, then the value of `CriteriaMinimumAge` is used. |
| `CriteriaProcessLimit` | null (none) | Specifies a minimum age, in days, before a request that completed with a warning is eligible for deletion. If this parameter value is null, then the value of `CriteriaMinimumAge` is used. |
| `CriteriaRetentionAgeCancel` | null (none) | Specifies a minimum age, in days, before a request that was canceled is eligible for deletion. If this parameter value is null, then the value of `CriteriaMinimumAge` is used. |
| `CriteriaProcessLimit` | null (none) | Specifies the maximum number of requests to be processed by the batch delete job. If null or zero (`0`), there is no limit. The default value is zero (`0`). |

## Examples

- Delete all purgeable requests:

```
batchDeleteSchedulerRequests()
```

- Delete all purgeable requests in the application `ESS_NATIVE_HOSTING_APP_LOGICAL_NAME`:

```
batchDeleteSchedulerRequests(desc='My purge for ESS
NativeApp',params={'CriteriaApplication':'ESS_NATIVE_HOSTING_APP_LOGICAL_NAME'})
```

- Delete all purgeable requests in the application `ESS_NATIVE_HOSTING_APP_LOGICAL_NAME` for which the job definition is `/oracle/apps/ess/custom/MyDef`:

```
batchDeleteSchedulerRequests(desc='My purge for ESS
NativeApp',params={'CriteriaApplication':'ESS_NATIVE_HOSTING_APP_LOGICAL_NAME','Cr
iteriaJobDefn':'JobDefinition://oracle/apps/ess/custom/MyDef'})
```

- Delete all purgeable requests in the application `ESS_NATIVE_HOSTING_APP_LOGICAL_NAME` for which the job type is `JAVA_TYPE`:

```
batchDeleteSchedulerRequests(desc='My purge for ESS NativeApp',
params={'CriteriaApplication':'ESS_NATIVE_HOSTING_APP_LOGICAL_NAME',
'CriteriaJobExecType':'JAVA_TYPE'})
```

- Submit batch delete job request using the schedule `/oracle/as/ess/essapp/custom/WeeklySch`:

```
batchDeleteSchedulerRequests(desc='Purge using WeekSch',schId='/oracle/as/ess/essapp/custom/WeeklySch')
```

### Related Shell Script

essBatchDeleteRequests

## Work Allocation Commands

Table 4-2 shows the WLST commands Oracle Enterprise Scheduler provides for work allocation.

***Table 4-2 Oracle Enterprise Scheduler Work Allocation Commands***

| Command | Description |
|---|---|
| addProcessorBinding | Bind a work assignment |
| createWorkshift | Create a custom workshift |
| deleteWorkAssignment | Delete work assignment metadata |
| deleteWorkshift | Delete workshift metadata |
| queryProcessorBindings | Query all processor bindings |
| queryWorkAssignments | Query work assignment metadata |
| queryWorkSchedules | Query schedule metadata for work allocation |
| queryWorkshifts | Query workshift metadata |
| removeProcessorBinding | Remove a processor binding |
| updateWorkshift | Update workshift resources |

## addProcessorBinding

Command Category: Application Management Commands

Use with WLST: Online

### Description

Binds a work assignment to the current Oracle Enterprise Scheduler processor.

### Syntax

```
addProcessorBinding(workAssignmentName,server=serverName,
[isExclusive=None][scope])
```

| Argument | Definition |
|---|---|
| *workAssignmentName* | Specifies the name of the work assignment to bind. |
| *server* | Specifies the name of the server to which to bind. |

| Argument | Definition |
|---|---|
| *isExclusive* | Optional. Specifies whether the binding is exclusive (`true`). The default value is `false`. |
| *scope* | Optional. Can be `server`, `group`, or `global`. The default value is `server`.<br>• `server` binds to the specified server.<br>• `group` binds to the processing group of the specified server.<br>• `global` binds globally for all current and future servers in the isolation group. |

### Examples

- Bind a work assignment to a processor:

```
addProcessorBinding('mypkg/MyTestWA', server='ess_server1')
```

- Bind a work assignment to a processor in exclusive mode.

```
addProcessorBinding('SimpleApp', server='ess_server1', isExclusive='true')
```

- Bind a work assignment to a processor globally.

```
addProcessorBinding('SimpleApp',server='ess_server1',scope='global')
```

## createWorkshift

Command Category: Application Management Commands

Use with WLST: Online

### Description

Creates a custom workshift that specifies a time window and the resources to be used during that window.

### Syntax

```
createWorkshift(workshiftName,alloc=allocation,
plsqlLimit=limit,asyncJavaLimit=limit,[description=None],
[schedName=None],[duration=None])
```

| Argument | Definition |
|---|---|
| *workshiftName* | Specifies the name of the workshift to create. |
| *alloc* | Specifies the maximum number of threads that can be allocated on each server instance. |
| *plsqlLimit* | Specifies the maximum number of PL/SQL jobs that can be active in the isolation group. Use `-1` to specify no limit. |
| *asyncJavaLimit* | Specifies the number of async Java jobs that can be active in the isolation group. Use -1 to specify no limit. |
| *description* | Optional. Specifies a description for the workshift. |

| Argument | Definition |
|---|---|
| *schedName* | Optional. Specifies the name of the schedule to use for the workshift.<br>• If the schedule name is specified, the `duration` argument is required.<br>• If the schedule name is *not* specified, the workshift is 24x7 |
| *duration* | Specifies the number of minutes the workshift is active each time the schedule runs. Required if the *schedName* argument is used. |

### Example

- Create a 24x7 workshift in a specific custom package:

```
createWorkshift('mypkg/SampleWorkshift', alloc=5, plsqlLimit=15,
asyncJavaLimit=10)
```

## deleteWorkAssignment

Command Category: Application Management Commands

Use with WLST: Online

### Description

Deletes an Oracle Enterprise Scheduler work assignment in the custom namespace.

**Note:**

A work assignment that is bound cannot be deleted.

### Syntax

```
deleteWorkAssignment(workAssignmentName)
```

| Argument | Definition |
|---|---|
| *workAssignmentName* | Specifies the name of the custom work assignment to delete. |

### Example

- Delete a custom work assignment:

```
deleteWorkAssignment('MyTestWA')
```

## deleteWorkshift

Command Category: Application Management Commands

Use with WLST: Online

### Description

Deletes an Oracle Enterprise Scheduler workshift in the custom namespace.

**Note:**

A workshift that is in use by an on-board tenant or any bound work assignment cannot be deleted.

### Syntax

```
deleteWorkshift(workshiftName)
```

| Argument | Definition |
|---|---|
| *workshiftName* | Specifies the name of the custom workshift to delete. |

### Example

- Delete a custom workshift:

```
deleteWorkshift('mypkg/TenantWorkshift1')
```

## queryProcessorBindings

Command Category: Application Management Commands

Use with WLST: Online

### Description

Gets processor bindings.

### Syntax

```
queryProcessorBindings(server=serverName)
```

| Argument | Definition |
|---|---|
| *server* | Specifies the name of the server to query. |

### Example

- Get all bindings:

```
queryProcessorBindings(server='ess_server1')
```

## queryWorkAssignments

Command Category: Application Management Commands

Use with WLST: Online

### Description

Gets all custom and internal work assignments. Prints the work assignment id. Details include workshift IDs and specialization.

### Syntax

```
queryWorkAssignments([name=None],[package=None],[options=None])
```

| Argument | Definition |
|---|---|
| *name* | Optional. Specifies a work assignment name. |
| *package* | Optional. Specifies a work assignment package. |
| *options* | Optional. Comma-separated list of one or more of the following options:<br>• `nameLike`<br>Query for a name that contains the specified name.<br>• `pkgLike`<br>Query for a package that contains the specified package.<br>• `details`<br>Print additional details. |

### Examples

- Print all work assignments:

```
queryWorkAssignments()
```

- Print details for work assignments whose name contains "Simple":

```
queryWorkAssignments(name='Simple', options='nameLike,details')
```

## queryWorkSchedules

Command Category: Application Management Commands

Use with WLST: Online

### Description

Gets all schedules for work allocation and prints the schedule ID. Details include recurrence, exclusions, inclusions.

### Syntax

```
queryWorkSchedules([name=None],[package=None],[options=None])
```

| Argument | Definition |
|---|---|
| *name* | Optional. Specifies the schedule name. |
| *package* | Optional. Specifies the schedule package. |
| *options* | Optional. Comma-separated list of one or more of the following options:<br>• `nameLike`<br>Query for a name that contains the specified name.<br>• `pkgLike`<br>Query for a package that contains the specified package.<br>• `details`<br>Print additional details. |

### Examples

- Get all schedules:

```
queryWorkSchedules()
```

- Get details for an hourly schedule:

```
queryWorkSchedules(name='Hourly', options='details')
```

## queryWorkshifts

Command Category: Application Management Commands

Use with WLST: Online

### Description

Gets all workshifts and prints the workshift ID. Details include thread allocation and async limits.

### Syntax

```
queryWorkshifts([name=None], [package=None], [options=None])
```

| Argument | Definition |
|---|---|
| *name* | Optional. Specifies the workshift name. |
| *package* | Optional. Specifies the workshift package. |
| *options* | Optional. Comma-separated list of one or more of the following options:<br>• `nameLike`<br>Query for a name that contains the specified name.<br>• `pkgLike`<br>Query for a package that contains the specified package.<br>• `details`<br>Print additional details. |

### Examples

- Get all workshifts:

```
queryWorkshifts()
```

- Get workshifts whose package contains '"test" and print the details:

```
queryWorkshifts(package='test', options='pkgLike,details')
```

## removeProcessorBinding

Command Category: Application Management Commands

Use with WLST: Online

### Description

Removes a local or global processor binding.

### Syntax

```
removeProcessorBinding(workAssignmentName, server=serverName)
```

| Argument | Definition |
|---|---|
| *workAssignmentName* | Specifies the name of the work assignment to unbind. |
| *server* | Specifies the name of the server from which to unbind. |

### Example

- Remove a binding:

```
removeProcessorBinding('mypkg/MyTestWA', server='ess_server1')
```

## updateWorkshift

Command Category: Application Management Commands

Use with WLST: Online

### Description

Update async limits and thread allocation for a custom workshift.

### Syntax

```
updateWorkshift(workshiftName,[alloc=None],[plsqlLimit=None],
[asyncJavaLimit=None])
```

| Argument | Definition |
|---|---|
| *workshiftName* | Specifies the name of the workshift to update. |
| *alloc* | Optional. Specifies the maximum local thread allocation. |
| *plsqlLimit* | Optional. Specifies the maximum number of PL/SQL jobs that can be active in the isolation group. Use -1 to specify that there no limit. |
| *asyncJavaLimit* | Optional. Specifies the maximum number of async Java jobs that can be active in the isolation group. Use -1 to specify that there is no limit. |

**Note:**

You must specify least one of the optional arguments.

### Examples

- Update the PL/SQL limit for the workshift `mypkg/SampleWorkshift`.

```
updateWorkshift('mypkg/SampleWorkshift', plsqlLimit=20)
```

- Update both async limits for the workshift `mypkg/SampleWorkshift`.

```
updateWorkshift('mypkg/SampleWorkshift', plsqlLimit=25, asyncJavaLimit=50)
```

# Oracle Enterprise Scheduler Diagnostic Dumps

Oracle Enterprise Scheduler provides a set of diagnostic dumps that facilitate the diagnosis of problems. These dumps are built on the Oracle Diagnostics Framework. Oracle Enterprise Scheduler problems are typically exposed when a request does not start or complete as expected. In such a scenario, the user can manually create Oracle Enterprise Scheduler diagnostic dumps and use the information in the dumps to help diagnose the problem. One way to easily create all Oracle Enterprise Scheduler diagnostic dumps is by creating an incident with a specific message ID, as described in Creating an XXX Oracle Enterprise Scheduler Incident using WLSTXXX.

The following are the Oracle Enterprise Scheduler diagnostic dump-related commands. They are described in The Dump Commands.

- executeDump
- `listDumps`
- `describeDump`

You control the behavior of the dump by specifying the dump name as an argument to the command. The valid dump names are described in Table 4-3 and examples of their use are shown in Dump Examples.

***Table 4-3 Oracle Enterprise Scheduler Diagnostic Dumps***

| Dump Name | Description |
|---|---|
| *ess.applications* | Dumps all registered Oracle Enterprise Scheduler applications on every server in the isolation group. An application that is registered on multiple servers appears multiple times in this dump.<br>For an application on a given server, you can check what version is registered, whether the application is active, and when it was last activated. |
| *ess.bindings* | Dumps current and inactive processor bindings for this instance, including the binding mode. An inactive binding that is completed has an expired schedule, and cannot become active again. For inactive bindings that are not completed, the dump shows the workshift that will next be active and when it will be active. This dump shows all completed bindings. |
| *ess.processor* | Dumps current processor information for this instance.<br>Dumps the processor state and active thread count, plus a list of enabled and deployed applications. The active thread count is the number of threads that are in use across all resources, including deactivated resources. For detailed information on work allocation, see dump *ess.workalloc*.<br>If an application is not processing requests, make sure the processor is running and the application is both deployed and enabled on the server. |
| *ess.requests* | Dumps detailed information about all requests in the process group that are in a non-terminal state. Includes core and extended system properties and CP parameters. |

*Table 4-3 (Cont.) Oracle Enterprise Scheduler Diagnostic Dumps*

| Dump Name | Description |
|---|---|
| *ess.servers* | Dumps server information, such as last check-in, for all active servers in the isolation group. Every server must check in at some interval (one minute by default).<br>If a server hasn't checked in for over one minute, it may have a problem. After a server hasn't checked in for some period of time, Oracle Enterprise Scheduler may assume the server is down and fail over the requests that were being processed on that server. |
| *ess.workalloc* | Dumps detailed information on work assignments, including maximum resources and current allocations. The `whereClause` shows the effective specialization for the work assignment, which accounts for work assignments bound in exclusive mode.<br>This dump may not show all completed bindings. Use *ess.bindings* to check if a binding is complete.<br>If there are no bindings, this dump shows information on the default work assignment. |
| *ess.workunits* | Dumps all work units for this process group. |

## The Dump Commands

The following example shows how to start WLST and connect to the server:

```
$MW_HOME/oracle_common/common/bin/wlst.sh
...
Initializing WebLogic Scripting Tool (WLST) ...

Welcome to WebLogic Server Administration Scripting Shell

Type help() for help on available commands

wls:/offline> connect('weblogic','welcome1','localhost:7001')
Connecting to t3://localhost:7001 with userid weblogic ...
Successfully connected to Admin Server 'AdminServer' that belongs to domain
'base_domain'.

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:/base_domain/serverConfig>
```

**Note:**

You can connect to the admin server or a managed server. However, if you connect to the admin server, you must specify the managed server to use with each dump command, for example:

```
executeDump(name="ess.applications",
appName="ESSAPP",server='ess_server1')
```

Once WLST is started and connected to the server, you can use the `executeDump`, `listDumps`, `describeDump` commands as shown in the following sections.

### executeDump

The `executeDump` command creates a diagnostic dump based on the dump name and the application name.

#### Syntax

```
executeDump(name="dump_name",appName="app_name",
[server="server_name"])
```

| Argument | Definition |
|---|---|
| *name* | The dump name from Table 4-3. |
| *appName* | The name of the application. |
| *server* | Optional. You can connect to the AdminServer or a managed server. If you connect to the AdminServer, you must use the `server` argument to specify the name of the managed server to use with each dump command. |

### listDumps

The `listDumps` command lists the available dumps for a given application.

#### Syntax

```
listDumps(appName="app_name")
```

| Argument | Definition |
|---|---|
| *appName* | The name of the application. |

### describeDump

Lists help for a specific dump type.

#### Syntax

describeDump(name="dump_name")

| Argument | Definition |
|---|---|
| *name* | The dump name from Table 4-3. |

## Dump Examples

The following examples show how to use the diagnostic dumps listed in Table 4-3.

- Dump all registered applications on every server in the isolation group. An application that is registered on more than one server appears multiple times in this dump. For an application on a given server, you can check what version is registered, whether the application is active, and when it was last activated.

  ```
  executeDump(name="ess.applications",appName="ESSAPP")
  ```

- Dump processor bindings for this instance, including the binding mode. An inactive binding that is completed has an expired schedule, and it does not become active again. For inactive bindings that are not completed, the dump shows the workshift that will be active next and when it will be active. This dump shows all completed bindings.

  ```
  executeDump(name="ess.bindings",appName="ESSAPP")
  ```

- Dump processor state and active thread count, plus list enabled and deployed applications. The active thread count is the number of threads that are in use across all resources, including deactivated resources. For more information about work allocation, see the entry for `ess.workalloc` in Table 4-3.

  If an application is not processing requests, make sure the processor is running and the application is both deployed and enabled on the server.

  ```
  executeDump(name="ess.processor",appName="ESSAPP")
  ```

- Dump detailed information about all requests in the process group that are in a non-terminal state. Includes core and extended system properties and CP parameters.

  ```
  executeDump(name="ess.requests",appName="ESSAPP")
  ```

- Dump all active servers in the isolation group. Every server must check in at an interval, one minute by default. If a server does not check in for over one minute, it may have a problem. If a server does not check in for some period of time, ESS may assume the server is down and fail over the requests that were being processed on that server.

  ```
  executeDump(name="ess.servers",appName="ESSAPP")
  ```

- Dump detailed information about work assignments, including maximum resources and current allocations. The `whereClause` information shows the effective specialization for the work assignment, which accounts for work assignments bound in exclusive mode.

  This dump may not show all completed bindings. To check whether a binding is complete, use the `ess.bindings` dump. If there are no bindings, this dump shows information about the default work assignment.

  ```
  executeDump(name="ess.workalloc",appName="ESSAPP")
  ```

- Dump all work units in the process group.

  ```
  executeDump(name="ess.workunits",appName="ESSAPP")
  ```

## Creating an Oracle Enterprise Scheduler Incident

Oracle Enterprise Scheduler provides a convenient way to create an incident with all Oracle Enterprise Scheduler diagnostic dumps, using the message id `ESS-99999`. For example:

```
createIncident(messageId="ESS-99999",appName="ESSAPP",
description="ESS incident with all dumps")
```

The incident files are located in the `ADRHome` directory, and you can use the `listADRHOmes()` command to list them.

Note that Oracle Enterprise Scheduler uses the following paths:

ADRBase = ${MW_HOME}/user_projects/domains/base_domain/servers/*SERVER*/adr

ADRHome = ${ADRBase}/diag/ofm/base_domain/*SERVER*

# Oracle Enterprise Scheduler Convenience Scripts

To simplify execution, wrapper shell scripts (`.sh` on Linux/Unix and `.cmd` on Windows) are provided to invoke the native WLST commands. The scripts set environment properties such as `CLASSPATH` before using `wlst.sh` to invoke the WLST native commands. All the scripts are available in the *$ORACLE_HOME*`/bin` directory.

The `.sh` and `.cmd` commands connect to a WLS server before executing the corresponding WLST command. By default, the server connection details are read from a file named `server.properties` in the `HOME` directory of the user running the command (typically `/home/`*UserId* on Linux/Unix and `C:\Documents and Settings\`*UserId* on Windows). You can override the default file by specifying an alternate file or by providing explicit values (host, port, user name, password) on the command line. The host, port, user name and password values specified on the command line take precedence over the corresponding values in the files.

The connection details must be specified in the following syntax in the `server.properties` file or in any alternate file provided on the command line:

```
ADMIN_SERVER_HOST=host.example.com
ADMIN_SERVER_PORT=7001
ADMIN_USER_NAME=weblogic
ADMIN_PASSWORD=weblogic1
ESS_SERVER_NAME=ess_server1
```

In order to comply with manageability checklist requirements, the password cannot be supplied as a command-line argument while invoking shell scripts. It can either be part of the `server.properties` file or entered interactively. You can use one of the following methods to automate interactive password submission:

- Write the WLS administrator password in a file, and then use it to redirect input to command. For example:

  `sh essManageRuntimeConfig.sh .... < input.txt`

- After the command finishes use "`<<EOF`". EOF can be any unique string other than the actual password. In the next line write the password, followed by EOF in the next line (or whatever terminating string you used after <<). For Example:

  ```
  sh essManageRuntimeConfig.sh ... <<EOF
  wls_admin_password
  EOF
  ```

## essManageRuntimeConfig

Manages Oracle Enterprise Scheduler runtime configuration parameters. The server connection parameters are specified using a file as described in Oracle Enterprise Scheduler Convenience Scripts. However, the default values can be overridden by explicitly specifying them on the command line or specifying an alternate file using the `-f` option. The admin server password for WLS has to be provided either in a file or entered interactively.

### Syntax

```
essManageRuntimeConfig.sh [-a | -m | -d] -A appname -t
parameter_type [-n name] [-v value] [-f filename] [-H hostname]
[-P port] [-u user] [-s server_name] [-h]
```

### Options

The following table lists and describes the command options.

| Option | Description |
|---|---|
| `-a`, `--add` | Adds a configuration parameter |
| `-m`, `--mod` | Modifies a configuration parameter |
| `-d`, `--del` | Deletes a configuration parameter. |
| `-A`, `--app` *`appname`* | Specifies the name of the hosting application. |
| `-t`, `--type` *`parameter_type`* | Specifies the type of the configuration parameter. Possible types are:<br>• `ESS` - for Oracle Enterprise Scheduler parameters<br>• `APP` - for user defined parameters |
| `-n`, `--name` *`name`* | Specifies the name of the configuration parameter. |
| `-v`, `--val` *`value`* | Specifies the value of the configuration parameter. |
| `-f`, `--file` *`filename`* | Specifies the name of the file that contains the server connection parameters. If not specified, the default file `$HOME/server.properties` is used. |
| `-H`, `--host` *`hostname`* | Specifies the hostname or IP address of the WLS server. |
| `-P`, `--port` *`port`* | Specifies the port number to connect. |
| `-u`, `--user` *`username`* | Specifies the user name. |
| `-s`, `--serv` *`server_name`* | Specifies the name of the server on which ESSAPP is running. |
| `-h` | Displays the command usage. |

### Associated Files

The following file is used with this command:

`server.properties:` Contains server connection information

### Exit Values

The command exits with the following possible values:

- 0: Success
- -1: Error

### Examples

- Add a user-defined parameter `foo` with value `bar` for the application `myapp`:

```
essManageRuntimeConfig.sh -a -A myapp -t APP -n foo -v bar
```

- Get the value of user-defined parameter `foo` for the application `myapp`:

```
essManageRuntimeConfig.sh -A myapp -t APP -n foo
```

- Get the list of all user-defined parameters for the application `myapp`:

```
essManageRuntimeConfig.sh -A myapp -t APP
```

- Modify the value of the user-defined parameter foo to `barone` for the application `myapp`:

```
essManageRuntimeConfig.sh -m -A myapp -t APP -n foo -v barone
```

- Delete the user-defined parameter `foo` for application `myapp`:

```
essManageRuntimeConfig.sh -d -t APP -A myapp -n foo
```

- Show all ESS parameters for `myapp`:

```
essManageRuntimeConfig.sh -t ESS -A myapp
```

### Related Native Command

Manage (Add/Delete/Modify/Get) Configuration Parameters

## essGetOutputContent

Retrieves the request log and output data files from the content store for all the specified request IDs. The server connection parameters are specified using a file as described in Oracle Enterprise Scheduler Convenience Scripts. However, the default values can be overridden by explicitly specifying them on the command line or specifying an alternate file using the `-f` option. The admin server password for WLS has to be provided in a file or entered interactively.

### Syntax

```
essGetOutputContent.sh [-t content_type] [-d dir] [-n lines] [-x
disp] [-f filename] [-H hostname] [-P port] [-u user] [-s
server_name] [-h] requestId1 ...
```

### Options

The following table lists and describes the command options.

| Option | Description |
|---|---|
| -t, `--content` *type* | Specifies one of the following types of content to get:<br>• `LOG`<br>• `OUTPUT`<br>• `TEXT_OUTPUT`<br>• `BINARY_OUTPUT`<br>If this option is not specified, both log and output content are retrieved. |

| Option | Description |
|---|---|
| `-d`, `--outDir` *`dir`* | Specifies the directory where content is stored. The default is the current directory. |
| `-n`, `--logLines` *`lines`* | Specifies the number of lines of log to retrieve. The default is 1000. |
| `-x`, `--logDisplay` *`disp`* | Specifies where log lines are presented. The valid values are `CONSOLE` or `FILE`. The default is `FILE`. |
| `-f`, `--file` *`filename`* | Specifies the name of the file that contains the server connection parameters. If not specified, the default file `$HOME/server.properties` is used. |
| `-H`, `--host` *`hostname`* | Specifies the hostname or IP address of the WLS server. |
| `-P`, `--port` *`port`* | Specifies the port number to connect. |
| `-u`, `--user` *`username`* | Specifies the user name. |
| `-s`, `--serv` *`server_name`* | Specifies the name of the server on which ESSAPP is running. |
| `-h` | Displays the command usage. |

### Associated Files

The following file is used with this command:

`server.properties:` Contains server connection information

### Exit Values

The command exits with the following possible values:

- 0: Success
- -1: Error

### Examples

- Get the request log for request ID `123`:

```
essGetOutputContent.sh -t LOG 123
```

- Get all of the output from request `123`:

```
essGetOutputContent.sh -t OUTPUT 123
```

- Get all the output of request `123` and save it in directory `/tmp`:

```
essGetOutputContent.sh -t OUTPUT -d "/tmp" 123
```

- Get the first 100 lines of the request log for request ID `123`:

```
essGetOutputContent.sh -t LOG -n 100 123
```

- Get the request log and output data for requests `123` and `124` and save it in `/tmp`:

```
essGetOutputContent.sh -d "/tmp" 123 124
```

### Related Native Command

Get Log and Output Content of a Request

## essQueryRequests

Search and list requests based upon hosting application name, state or elapsed time of execution. This command can be used to find long running requests. The server connection parameters are specified using a file as described in Oracle Enterprise Scheduler Convenience Scripts. However, the default values can be overridden by explicitly specifying them on the command line or specifying an alternate file using the `-f` option. The admin server password for WLS has to be provided in a file or entered interactively.

### Syntax

```
essQueryRequests.sh [-a app_name] [-S state] [-d days] [-n
hours] [-m minutes] [-f filename] [-H hostname] [-P port] [-u
user] [-s server_name] [-h]
```

### Options

The following table lists and describes the command options.

| Option | Description |
|---|---|
| `-a`, `--appName` *`app_name`* | Specifies the hosting application name. |
| `-S`, `--state` *`state`* | Specifies one of the following request states:<br>• `WAIT`<br>• `READY`<br>• `RUNNING`<br>• `COMPLETED`<br>• `BLOCKED`<br>• `HOLD`<br>• `CANCELLING`<br>• `EXPIRED`<br>• `CANCELLED`<br>• `ERROR`<br>• `WARNING`<br>• `SUCCEEDED`<br>• `PAUSED`<br>• `PENDING_VALIDATION`<br>• `VALIDATION_FAILED`<br>• `SCHEDULE_ENDED`<br>• `FINISHED`<br>• `ERROR_AUTO_RETRY`<br>• `ERROR_MANUAL_RECOVERY` |
| `-d`, `--days` *`days`* | Specifies the running time in days. |
| `-n`, `--hours` *`hours`* | Specifies the running time in hours. |
| `-m`, `--minutes` *`minutes`* | Specifies the running time in minutes. |

| Option | Description |
|---|---|
| `-f`, `--file` *`filename`* | Specifies the name of the file that contains the server connection parameters. If not specified, the default file `$HOME/server.properties` is used. |
| `-H`, `--host` *`hostname`* | Specifies the hostname or IP address of the WLS server. |
| `-P`, `--port` *`port`* | Specifies the port number to connect. |
| `-u`, `--user` *`username`* | Specifies the user name. |
| `-s`, `--serv` *`server_name`* | Specifies the name of the server on which ESSAPP is running. |
| `-h` | Displays the command usage. |

### Associated Files

The following file is used with this command:

`server.properties:` Contains server connection information

### Exit Values

The command exits with the following possible values:

- 0: Success
- -1: Error

### Examples

- Get all running requests:

  ```
  essQueryRequests.sh -S RUNNING
  ```

- Get all cancelled requests:

  ```
  eessQueryRequests.sh -S CANCELLED
  ```

- Get all requests running for more than two days:

  ```
  essQueryRequests.sh -d 2
  ```

- Get all requests running for more than 10 hours:

  ```
  essQueryRequests.sh -n 10
  ```

- Get all requests running for the application `myapp`:

  ```
  essQueryRequests.sh -a myapp
  ```

- Get all requests in the CANCELLING state for the application `myapp`:

  ```
  essQueryRequests.sh -a myapp -S CANCELLING
  ```

- Get all requests that have been running for more than 2.5 days for the application `myapp`:

  ```
  essQueryRequests.sh -a myapp -d 2 -n 12
  ```

### Related Native Command

Search and List Requests

## essManageRequests

Cancel, recover, or complete one or more requests given their request IDs. The server connection parameters are specified using a file as described in Oracle Enterprise Scheduler Convenience Scripts. However, the default values can be overridden by explicitly specifying them on the command line or specifying an alternate file using the `-f` option. The admin server password for WLS has to be provided in a file or entered interactively.

### Syntax

```
essManageRequests.sh [-c] [-r] [-C] [-g] [-S status] [-M
message] [-f filename] [-H hostname] [-P port] [-u user] [-s
server_name] [-h] requestId1 ...
```

### Options

The following table lists and describes the command options.

| Option | Description |
|---|---|
| `-c`, `--cancel` | Cancels request. |
| `-r`, `--recover` | Recovers a request |
| `-C`, `--complete` | Marks a request as completed. |
| `-g`, `--getdetail` | Gets a request's details. |
| `-S`, `--asyncStatus` *`status`* | When marking request as complete, specifies one of the following final disposition statuses:<br>• `SUCCESS`<br>• `PAUSE`<br>• `WARNING`<br>• `ERROR`<br>• `CANCEL`<br>• `UPDATE`<br>• `BIZ_ERROR`<br>• `ERROR_MANUAL_RECOVERY` |
| `-M`, `--statusMessage` *`message`* | Specifies a qualifying message that describes the operation when marking request as complete. |
| `-f`, `--file` *`filename`* | Specifies the name of the file that contains the server connection parameters. If not specified, the default file `$HOME/server.properties` is used. |
| `-H`, `--host` *`hostname`* | Specifies the hostname or IP address of the WLS server. |
| `-P`, `--port` *`port`* | Specifies the port number to connect. |
| `-u`, `--user` *`username`* | Specifies the user name. |

| Option | Description |
|---|---|
| `-s, --serv` *`server_name`* | Specifies the name of the server on which ESSAPP is running. |
| `-h` | Displays the command usage. |

### Associated Files

The following file is used with this command:

`server.properties:` Contains server connection information

### Exit Values

The command exits with the following possible values:

- 0: Success
- -1: Error

### Examples

- Cancel requests 123 and 124:

```
essManageRequests.sh -c 123 124
```

- Recover request 123:

```
essManageRequests.sh -r 123
```

- Complete request 123:

```
essManageRequests.sh -C -S ERROR -M "Completed by Admin" 123
```

### Related Native Command

Manage Requests

## essManageServer

Start, stop or get the status of the Oracle Enterprise Scheduler application running on the server. Starting the Oracle Enterprise Scheduler application means to start the Oracle Enterprise Scheduler processor thread so that request processing can start. Stopping Oracle Enterprise Scheduler means to stop or quiesce the Oracle Enterprise Scheduler processor so that no new requests are processed. The server connection parameters are specified using a file as described in Oracle Enterprise Scheduler Convenience Scripts. However, the default values can be overridden by explicitly specifying them on the command line or specifying an alternate file using the `-f` option. The admin server password for WLS has to be provided in a file or entered interactively.

### Syntax

```
essManageServer.sh [-start] [stop] [-status] [-f filename] [-H
hostname] [-P port] [-u user] [-s server_name] [-h]
```

### Options

The following table lists and describes the command options.

| Option | Description |
|---|---|
| `-n, --start` | Starts the Oracle Enterprise Scheduler processor thread. |
| `-x, --stop` | Stops the Oracle Enterprise Scheduler processor thread. |
| `-S, --status` | Displays the current status of the Oracle Enterprise Scheduler processor thread. |
| `-f, --file` *`filename`* | Specifies the name of the file that contains the server connection parameters. If not specified, the default file `$HOME/server.properties` is used. |
| `-H, --host` *`hostname`* | Specifies the hostname or IP address of the WLS server. |
| `-P, --port` *`port`* | Specifies the port number to connect. |
| `-u, --user` *`username`* | Specifies the user name. |
| `-s, --serv` *`server_name`* | Specifies the name of the server on which ESSAPP is running. |
| `-h` | Displays the command usage. |

### Associated Files

The following file is used with this command:

`server.properties:` Contains server connection information

### Exit Values

The command exits with the following possible values:

- 0: Success
- -1: Error

### Examples

- Stop Oracle Enterprise Scheduler processor:

```
essManageServer.sh --stop
```

- Get the current state of the Oracle Enterprise Scheduler processor:

```
essManageServer.sh --status
```

### Related Native Command

Manage Oracle Enterprise Scheduler Servers

## essSubmitRequest

Submit a job request to ESSAPP for execution. It is assumed that the job definition exists in the MDS. The user can specify the job execution schedule in the following four ways:

- Immediate: The job is scheduled for immediate one-time execution.

- Once: The job is scheduled to run once at a fixed time specified by the user.
- Schedule: The name of a predefined schedule (in MDS) is provided and the job runs on that schedule.
- Recurring: The schedule can be created in an ad-hoc manner by providing relevant arguments through this command. This schedule is used for the request submission and it is stored in the MDS for future use. The storage location is based on the schedule name provided.

If no scheduling method is specified, immediate execution is used.

**Note:**

The job definition must be predefined, while the schedule may be defined in an ad-hoc manner using this command.

The `jobName` and `schName` options must be fully qualified names that includes the package name. If they are not fully qualified, the following package is used: `/oracle/apps/ess/custom/`.

The server connection parameters are specified using a file as described in Oracle Enterprise Scheduler Convenience Scripts. However, the default values can be overridden by explicitly specifying them on the command line or specifying an alternate file using the `-f` option. The admin server password for WLS has to be provided in a file or entered interactively.

### Syntax

```
essSubmitRequest.sh appName jobName [-n reqNote] [-I | -O time |
-R params | -S schName] [-f filename] [-H hostname] [-P port] [-
u user] [-s server_name] [-h] [prop=value ...]
```

### Options

The following table lists and describes the command options.

| Option | Description |
|---|---|
| `appName` | Names the job's hosting application. |
| `jobName` | Names the job definition to be used for the current request. |
| `-n`, `--note reqNote` | Specifies a short description for the job request |
| `-S`, --schName *`schName`* | Names the predefined schedule definition. |
| `-A`, `--RschName` *`name`* | Names the schedule if you define the schedule using `-R` option. |
| `-C`, `--schDesc` *`desc`* | Specifies a description of the ad-hoc schedule being defined. The default value is "Custom Ad-Hoc Schedule". |
| `-I`, `--immediate` | Specifies that the job should be executed immediately. This is used as the default if no other schedule is specified. |

| Option | Description |
|---|---|
| -O, --o_time *time* | Specifies the time for one time execution. Format: HH:MM:SS:DD:MM:YYYY. |
| -R, --recurring | Specifies that the schedule recurs. |
| -b, --begin_time *time* | Specifies the start time for a recurring schedule. Format: HH:MM:SS:DD:MM:YYYY . |
| -F, --frequency *value* | Specifies the frequency of recurrence. Valid values: [SECOND,MINUTE,HOUR,DAY,WEEK,MONTH,YEAR] |
| -N, --freqNum *num* | Specifies an integer value that represents the frequency repeat interval. |
| -c, --count *num* | Specifies an integer value that is the maximum number of repetitions. |
| -e, --end_time *time* | Specifies the recurrence end time. Format: HH:MM:SS:DD:MM:YYYY. |
| -m, --month *month* | Defines the month of the year if the frequency is YEAR. Valid values are: [1-12]. |
| -w, --week *week* | Specifies the week of a month. Valid values are: [1-6, LAST]. |
| -d, --day *day* | Specifies the day of a week. Valid values are: [1-7], where 1 represents Monday. |
| -D, --date *date* | Specifies a specific day of month. Valid values are: [1-31, LAST]. |
| *prop=value* | Specifies other request parameters/properties specified as: prop1=*val1* *prop2=val2* and so on. These values should be added as the last options in the command. |
| -f, --file *filename* | Specifies the name of the file that contains the server connection parameters. If not specified, the default file $HOME/server.properties is used. |
| -H, --host *hostname* | Specifies the hostname or IP address of the WLS server. |
| -P, --port *port* | Specifies the port number to connect. |
| -u, --user *username* | Specifies the user name. |
| -s, --serv *server_name* | Specifies the name of the server on which ESSAPP is running. |
| -h | Displays the command usage. |

### Associated Files

The following file is used with this command:

server.properties: Contains server connection information

### Exit Values

The command exits with the following possible values:

- 0: Success
- -1: Error

### Examples

The following values apply to all of the examples:

`HOSTING_APP` = `EssDemoApp`

`SCHEDULE_NAME`= `/oracle/apps/ess/custom/MySch`

`JOB_DEF_NAME`= `/oracle/apps/ess/demopackage/BasicJavaJob`

and server connection properties are defined in the `~/server.properties` file.

---

**Note:**

The `[prop-value ...]` values should be added as the last options in the command.

---

- Submit a request for immediate job execution (one-time only).:

```
essSubmitRequest.sh HOSTING_APP JOB_DEF_NAME
```

- Submit a request for one-time job execution at a specified time.:

```
essSubmitRequest.sh HOSTING_APP JOB_DEF_NAME -O 21:10:30:03:05:2012
```

- Submit a job request with a recurring schedule. The recurrence is defined as follows:
  - Execution recurs every 2 minutes
  - Starts at 21:10:30 on 03 May 2012
  - Completes three iterations
  - Additional request parameters are: `eqName=test` and `PRIORITY=10`. This recurrence will persisted in the MDS repository for future use.

```
essSubmitRequest.sh HOSTING_APP JOB_DEF_NAME -R -A SCHEDULE_NAME -F MINUTE -N 2 -
c 3 -b 21:10:30:03:05:2012 reqName=test PRIORITY=10
```

- Submit a job request using a predefined schedule:

```
essSubmitRequest.sh HOSTING_APP JOB_DEF_NAME -S SCHEDULE_NAME
```

- Submit a job request with a monthly recurring schedule. The description for the request is "Request with monthly schedule". The recurring schedule name and description are provided by `-A` and `-C` options. The recurrence is: Execution recurs every 2 months, on Thursday of the 5th week (if applicable), starting at 21:10:30 on 03 May 2012, until 21:10:30 on 04 May 2013:

```
essSubmitRequest.sh HOSTING_APP JOB_DEF_NAME -n "Request with monthly schedule" -
R -A SCHEDULE_NAME -C Monthly_schedule -F MONTH -N 2 -b 21:10:30:03:05:2012 -w 5 -
d 4 -e 21:10:30:04:05:2013
```

- Submit a job request with a yearly recurring schedule. The description for the request is "Request with yearly schedule". The recurring schedule name and description are provided by the `-A` and `-C` options. The recurrence is: Execution will recur every 3 years, on the last day of May, starting at 21:10:30 on 03 May 2012, for 5 iterations:

```
essSubmitRequest.sh HOSTING_APP JOB_DEF_NAME -n "Request with yearly schedule" -R
-A SCHEDULE_NAME-C "Yearly_schedule" -F YEAR -N 3 -b 21:10:30:03:05:2012 -m 5 -D
LAST -c 5
```

- Submit a job request with an hourly recurring schedule. Recurrence is: Execution repeats every hour starting at the current time.:

```
essQueryRequests.sh -a myapp -d 2 -n 12
```

### Related Native Command

Submit Job Requests to Oracle Enterprise Scheduler

## essManageJobDefn

Manage (show, create, delete, customize, update) an Oracle Enterprise Scheduler job definition.

The `show` option (`-S`) displays the list of job definitions that are part of a particular application. If a job name is specified it only shows the details of this job definition.The `create` option (`-N`) creates a new job definition in the namespace of the relevant application (supplied as a parameter) in MDS.The `delete` option (`-D`) deletes a job definition from the MDS.The `customize` option (`-C`) modifies the customizable properties of an existing job definition.The `update` option (`-U`) updates an existing job definition with the desired property values (existing parameters persist if not overridden).

---

**Note:**

- For job definition creation, deletion and update, the default package path `/oracle/apps/ess/custom/` is always prepended to the specified job name.
- You can only create, delete, or update the job definitions present in the `/oracle/apps/ess/custom` namespace. That means that prepackaged job definitions cannot be modified.
- You cannot customize job definitions present in the customer namespace `/oracle/apps/ess/custom`. Use the update option to modify those definitions.
- For the `jobType` specification, if you do not specify a fully qualified name, the default package `/oracle/as/ess/core/` is prepended to the name.

---

The server connection parameters are specified using a file as described in Oracle Enterprise Scheduler Convenience Scripts. However, the default values can be overridden by explicitly specifying them on the command line or specifying an alternate file using the `-f` option. The admin server password for WLS has to be provided in a file or entered interactively.

### Syntax

```
essManageJobDefn.sh appName [-j jobName] [-t jobType] [-d desc]
-S | -N | -D | -C | -U [-f filename] [-H hostname] [-P port] [-u
user] [-s server_name] [-h] [prop=value ...]
```

### Options

The following table lists and describes the command options.

| Option | Description |
|---|---|
| *appName* | Specifies the logical name of the hosting application for a job. |
| -j, --jobName *jobDefn* | Specifies the name of the job definition. For the create, delete, and update operations, the default package /oracle/apps/ess/custom/ is prepended to the specified name. |
| -t, --jobType *jobType* | Specifies the name of the predefined job type to be used for creating the job definition. If the name does not contain package path, the following path is prepended to it: /oracle/as/ess/core/. |
| -d, --desc *description* | Specifies a description of the job definition |
| -S, --show | Displays the list of job definitions that are part of a particular application. If a job name is specified it shows the details of only that job definition. Used with *appName* and -j. |
| -N, --create | Creates a new job definition. Used with *appName*, -j, -t, -d and *prop*=*value*. |
| -D, --delete | Deletes an existing job definition. Used with *appName* and -j. |
| -C, --customize | Modifies customizable properties of the job definition. Properties can be:<br>• SYS_retries<br>• SYS_priority<br>• SYS_requestCategory<br>• SYS_request_timeout<br>• enableTrace<br>• enableTimeStatistics<br>Used with *appName*, -j option, *prop*=*value*. |
| -U, --update | Updates an existing job definition. This option replaces the current definition with a new one in which existing parameters persist if not overridden. Used with *appName*, -j and either -d or *prop*=*value*. |
| *prop*=*value* | Specifies other request parameters/properties specified as: *prop1=val1 prop2=val2* and so on. |

| Option | Description |
|---|---|
| `-f, --file filename` | Specifies the name of the file that contains the server connection parameters. If not specified, the default file `$HOME/server.properties` is used. |
| `-H, --host hostname` | Specifies the hostname or IP address of the WLS server. |
| `-P, --port port` | Specifies the port number to connect. |
| `-u, --user username` | Specifies the user name. |
| `-s, --serv server_name` | Specifies the name of the server on which ESSAPP is running. |
| `-h` | Displays the command usage. |

### Associated Files

The following file is used with this command:

`server.properties:` Contains server connection information

### Exit Values

The command exits with the following possible values:

- 0: Success
- -1: Error

### Examples

The following values apply to all of the examples:

`HOSTING_APP` = `ESS_NATIVE_HOSTING_APP_LOGICAL_NAME`

`JOB_DEF_PREDEF`= `/oracle/apps/ess/TestJob`

`JOB_DEF_NAME` = `TestJob_wlst`

`JOB_TYPE_NAME` = `JavaJobType`

`JOB_DESC` = `My WLST Test Defn`

and server connection properties are defined in the `~/server.properties` file.

- Show all job definitions present in namespace of `HOSTING_APP`:

  ```
  essManageJobDefn.sh HOSTING_APP -S
  ```

- Show details of the job definition `JOB_DEF_PREDEF` part of `HOSTING_APP`:

  ```
  essManageJobDefn.sh HOSTING_APP -S -j JOB_DEF_PREDEF
  ```

- Create a new job definition named `/oracle/apps/ess/custom/TestJob_wlst` in the namespace of `HOSTING_APP` with the job type `/oracle/as/ess/core/JavaJobType`:

  ```
  essManageJobDefn.sh HOSTING_APP -N -j JOB_DEF_NAME -t JOB_TYPE_NAME
  ```

- Create a new job definition named `/oracle/apps/ess/custom/TestJob_wlst` in namespace of `HOSTING_APP` with the job type `/oracle/as/ess/core/JavaJobType` and with the description `JOB_DESC`:

```
essManageJobDefn.sh HOSTING_APP -N -j JOB_DEF_NAME -t JOB_TYPE_NAME -d JOB_DESC
```

- Create a new job definition named `/oracle/apps/ess/custom/TestJob_wlst` in the namespace of `HOSTING_APP` with the job type `/oracle/as/ess/core/JavaJobType` and with the description `JOB_DESC` and the properties `SYS_retries` and `myParam`:

```
essManageJobDefn.sh HOSTING_APP -N -j JOB_DEF_NAME -t JOB_TYPE_NAME -d JOB_DESC
SYS_retries=1 myParam=xyz
```

- Update the job definition `/oracle/apps/ess/custom+JOB_DEF_NAME` with the new description `JOB_DESC`:

```
essManageJobDefn.sh HOSTING_APP -U -j JOB_DEF_NAME -d JOB_DESC
```

- Update the job definition `/oracle/apps/ess/custom+JOB_DEF_NAME` with the properties `SYS_retries` and `myParam`:

```
essManageJobDefn.sh HOSTING_APP -U -j JOB_DEF_NAME SYS_retries=1 myParam=xyz
```

- Delete the job definition `/oracle/apps/ess/custom+JOB_DEF_NAME` in `HOSTING_APP`:

```
essManageJobDefn.sh HOSTING_APP -D -j JOB_DEF_NAME
```

- Customize the prepackaged job definition `JOB_DEF_PREDEF` in `HOSTING_APP` with the customizable properties SYS_retries & SYS_priority.:

```
essManageJobDefn.sh HOSTING_APP -C -j JOB_DEF_PREDEF SYS_retries=2 SYS_priority=3
```

### Related Native Command

Manage Oracle Enterprise Scheduler Job Definitions

## essManageSchedule

Manage (show, create, delete, customize and update) an Oracle Enterprise Scheduler schedule definition.

The `show` option (-S) displays the list of schedule definitions that are part of a particular application. If a schedule name is specified, it one shows the details for that schedule definition.

The `create` option (-C) creates a new schedule definition in the namespace of the relevant application (supplied as a parameter) in the MDS.

The `delete` option (`-D`) deletes the specified schedule definition from the MDS.

The `update` option (`-U`) updates an existing schedule definition with the specified property values (existing parameters persist if not overridden).

The server connection parameters are specified using a file as described in Oracle Enterprise Scheduler Convenience Scripts. However, the default values can be overridden by explicitly specifying them on the command line or specifying an alternate file using the `-f` option. The admin server password for WLS has to be provided in a file or entered interactively.

**Note:**

The default package name: `/oracle/apps/ess/custom/` is always prepended to the schedule name specified with the -n option.

### Syntax

```
essManageSchedule.sh appName [-n schName] [-d desc] -S | -C | -D
| -U -F value [-I interval] [-b begin_time] [-c count | -e
end_time][-f filename] [-H hostname] [-P port] [-u user] [-s
server_name]
```

### Options

The following table lists and describes the command options.

| Option | Description |
|---|---|
| *appName* | Specifies the logical name of the hosting application. |
| -n, --schName *schName* | Specifies the name of the schedule definition. For the create, delete, and update operations, the default package `/oracle/apps/ess/custom/` is prepended to the specified name. |
| -d, --schDesc *description* | Specifies a description of the schedule definition |
| -S, --show | Displays the list of schedule definitions that are part of a particular application. If a schedule name is specified it shows the details of only that schedule definition. Used with *appName*, -n and -d. |
| -C, --create | Creates a new schedule definition. Used with *appName*, -j, -t, -d and *prop=value*. |
| -D, --delete | Deletes an existing schedule definition. Used with *appName* and -n. |
| -U, --update | Updates an existing schedule definition. This option replaces the current definition with a new one in which existing parameters persist if not overridden. Used with *appName*, -n and either -d or -F. |
| -b, --begin_time *time* | Specifies the recurrence start time. Format: HH:MM:SS:DD:MM:YYYY |
| -F, --frequency *value* | Specifies the frequency of recurrence. Valid values: SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, YEAR<br>Possible options:<br>-F *value* [-I *interval*] [-b *begin_time*] [-c *count* \| -e *end_time*] (if *value* is SECOND, MINUTE, HOUR, DAY, WEEK otherwise, one of the following)<br>-F MONTH [{-w *week* -d *day*} \| {-D *date*}] [-b *begin_time*] [-c *count* \| -e *end_time*]<br>-F YEAR [-m *month* ({-w *week* -d *day*} \| {-D *date*})] [-b *begin_time*] [-c *count* \| -e *end_time*] |

| Option | Description |
|---|---|
| `-I`, `--interval` *num* | Specifies an integer value that is used as the repeat interval for the `-F` option. |
| `-c`, `--count` *num* | Specifies an integer value that is the maximum number of repetitions. |
| `-e`, `--end_time` *time* | Specifies the recurrence end time. Format: `HH:MM:SS:DD:MM:YYYY`. |
| `-m`, `--month` *month* | Defines the month of the year if the frequency is YEAR. Valid values are: [`1-12`]. |
| `-f`, `--file` *filename* | Specifies the name of the file that contains the server connection parameters. If not specified, the default file `$HOME/server.properties` is used. |
| `-H`, `--host` *hostname* | Specifies the hostname or IP address of the WLS server. |
| `-P`, `--port` *port* | Specifies the port number to connect. |
| `-u`, `--user` *username* | Specifies the user name. |
| `-s`, `--serv` *server_name* | Specifies the name of the server on which ESSAPP is running. |
| `-h` | Displays the command usage. |

### Associated Files

The following file is used with this command:

`server.properties:` Contains server connection information

### Exit Values

The command exits with the following possible values:

- 0: Success
- -1: Error

### Examples

The following values apply to all of the examples:

`HOSTING_APP` = `ESS_NATIVE_HOSTING_APP_LOGICAL_NAME`

`PREDEF_SCHEDULE` = `/oracle/apps/ess/demo/seeded/MyPredefinedSch`

`SCHEDULE_NAME` = `MyWlstSchedule`

`SCH_DESC` = `Description for WLST test schedule`

and server connection properties are defined in the `~/server.properties` file.

- Show all schedule definitions present in namespace of `HOSTING_APP`:

  ```
  essManageSchedule.sh HOSTING_APP -S
  ```

- Show details of the schedule definition `PREDEF_SCHEDULE` of `HOSTING_APP`:

```
essManageSchedule.sh HOSTING_APP -S -n PREDEF_SCHEDULE
```

- Create a new recurring schedule definition named `/oracle/apps/ess/custom/MyWlstSchedule` in the namespace of `HOSTING_APP` with the description `SCH_DESC`. The recurrence is: Occur every 2 minutes, starting at 21:10:30 on 03 May 2012, for 3 iterations:

  ```
  essManageSchedule.sh HOSTING_APP -C -n SCHEDULE_NAME -d SCH_DESC -F MINUTE -I 2 -
  c 3 -b 21:10:30:03:05:12
  ```

- Create a new recurring schedule definition named `/oracle/apps/ess/custom/MyWlstSchedule` in the namespace of `HOSTING_APP`. The recurrence is: Occur every 2 months, on Thursday of the 5th week (if applicable), starting at 21:10:30 on 03 May 2012, and running until 21:10:30 on 04 May 2013:

  ```
  essManageSchedule.sh HOSTING_APP -C -n SCHEDULE_NAME -F MONTH -I 2 -b
  21:10:30:03:05:12 -w 5 -x 4 -e 21:10:30:04:05:13
  ```

- Create a new recurring schedule definition named `/oracle/apps/ess/custom/MyWlstSchedule` in the namespace of `HOSTING_APP`. The recurrence is: Occur every 3 years, on the last day of May, starting at 21:10:30 on 03 May 2012, for 5 iterations:

  ```
  essManageSchedule.sh HOSTING_APP -C -n SCHEDULE_NAME -F YEAR -I 3 -b
  21:10:30:03:05:12 -m 5 -y LAST -c 5
  ```

- Create a new recurring schedule definition named `/oracle/apps/ess/custom/MyWlstSchedule` in the namespace of `HOSTING_APP`. The recurrence is: Occur every hour, starting from the current time:

  ```
  essManageSchedule.sh HOSTING_APP -C -n SCHEDULE_NAME -F HOUR
  ```

- Update the schedule definition `/oracle/apps/ess/custom+SCHEDULE_NAME` with the new description `SCH_DESC`. Recurrence remains unchanged:

  ```
  essManageSchedule.sh HOSTING_APP -U -n SCHEDULE_NAME -d SCH_DESC
  ```

- Update the schedule definition `/oracle/apps/ess/custom+SCHEDULE_NAME` with the following new recurrence - occur every minute:

  ```
  essManageSchedule.sh HOSTING_APP -U -n SCHEDULE_NAME -F MINUTE
  ```

- Delete the schedule definition `/oracle/apps/ess/custom+SCHEDULE_NAME` in `HOSTING_APP`:

  ```
  essManageSchedule.sh HOSTING_APP -D -n SCHEDULE_NAME
  ```

### Related Native Command

Manage Oracle Enterprise Scheduler Schedule Definitions

## essBatchDeleteRequests

Submits a request for a batch delete job. User can either use a pre-defined schedule (in MDS) or specify a one-time execution time. If neither of these is specified, the request starts immediately.

The request parameters are used to specify the delete criteria. When the batch delete job runs, it determines which completed absolute parent and instance parent requests satisfy the delete criteria specified for that batch job request and deletes the request hierarchy for those requests.

For detailed list and description of valid batch delete parameters, see Batch Delete Parameters.

The server connection parameters are specified using a file as described in Oracle Enterprise Scheduler Convenience Scripts. However, the default values can be overridden by explicitly specifying them on the command line or specifying an alternate file using the `-f` option. The admin server password for WLS has to be provided in a file or entered interactively.

### Syntax

```
essBatchDeleteRequests.sh [-d desc] [-S schId] [-b beginTime] [-
e endTime] [-f filename] [-H hostname] [-P port] [-u user] [-s
server_name] [-h] [prop=value ...]
```

### Options

The following table lists and describes the command options.

| Option | Description |
|---|---|
| `-d`, `--desc` *desc* | Specifies a description for the batch delete request. |
| `-S`, `--schId` *schId* | Specifies the fully qualified name of the schedule to use. The default package (used if the package is not specified) is `/oracle/as/ess/essapp/custom/`. |
| `-b`, `--begin` *beginTime* | Specifies the time at which the request is to be started. Format: `HH:MM:SS:DD:MM:YYYY` |
| `-e`, `--end` *endTime* | Specifies the time past which the request should not run. Format: `HH:MM:SS:DD:MM:YYYY` . |
| *prop=value* | Specifies other request parameters/properties specified as: *prop1=val1 prop2=val2* and so on. See Batch Delete Parameters for a list of the parameters. |
| `-f`, `--file` *filename* | Specifies the name of the file that contains the server connection parameters. If not specified, the default file `$HOME/server.properties` is used. |
| `-H`, `--host` *hostname* | Specifies the hostname or IP address of the WLS server. |
| `-P`, `--port` *port* | Specifies the port number to connect. |
| `-u`, `--user` *username* | Specifies the user name. |
| `-s`, `--serv` *server_name* | Specifies the name of the server on which ESSAPP is running. |
| `-h` | Displays the command usage. |

### Associated Files

The following file is used with this command:

`server.properties:` Contains server connection information

### Exit Values

The command exits with the following possible values:

- 0: Success
- -1: Error

### Examples

The following examples assume that server connection properties are defined in the `~/server.properties` file.

- Delete all purgeable requests.:

```
essBatchDeleteRequests.sh
```

- Delete all purgeable requests in the application `ESS_NATIVE_HOSTING_APP_LOGICAL_NAME`:

```
essBatchDeleteRequests.sh --desc "My purge for ESS NativeApp"
CriteriaApplication=ESS_NATIVE_HOSTING_APP_LOGICAL_NAME
```

- Delete all purgeable requests in application `ESS_NATIVE_HOSTING_APP_LOGICAL_NAME`, for which the job definition is `JobDefinition://oracle/apps/ess/custom/MyDef`:

```
essBatchDeleteRequests.sh --desc "My purge for ESS NativeApp"
CriteriaApplication=ESS_NATIVE_HOSTING_APP_LOGICAL_NAMEsCriteriaJobDefn=JobDefinit
ion://oracle/apps/ess/custom/MyDef
```

- Submit batch delete job request using the schedule `/oracle/as/ess/essapp/custom/WeeklySch`:

```
essBatchDeleteRequests.sh --desc "Purge using WeekSch" --schId /oracle/as/ess/
essapp/custom/WeeklySch
```

- Delete all purgeable requests submitted by the user `weblogic`:

```
essBatchDeleteRequests.sh CriteriaSubmitUser=weblogic
```

- Delete at most ten purgeable requests:

```
essBatchDeleteRequests.sh CriteriaProcessLimit=10
```

- Delete purgeable requests whose job execution type is `JAVA_TYPE`:

```
essBatchDeleteRequests.sh CriteriaJobExecType=JAVA_TYPE
```

- Delete purgeable requests that completed at five or more days ago:

```
essBatchDeleteRequests.sh CriteriaMinimumAge=5
```

- Delete purgeable requests, but retain requests that completed successfully in the past five days:

```
essBatchDeleteRequests.sh CriteriaRetentionAgeSuccess=5
```

- Delete purgeable requests that belong to the product named "DemoProduct":

```
essBatchDeleteRequests.sh CriteriaProduct=DemoProduct
```

### Related Native Command

The batchDeleteSchedulerRequest Command

# 5

# Oracle Managed File Transfer Custom WLST Commands

This chapter summarizes WLST (Oracle WebLogic Scripting Tool) commands that perform Oracle Managed File Transfer (MFT) operations.

This chapter includes the following sections:



For general information about WLST, see the *WLST Command Reference for WebLogic Server*.

For general information about MFT, see *Using Oracle Managed File Transfer*.

## Overview of MFT WLST Command Categories

MFT WLST commands are divided into the following categories.

***Table 5-1 MFT WLST Command Categories***

| Command Category | Description |
|---|---|
| MFT Artifact Management Commands | Perform these operations on source, transfer, and target artifacts: enable, disable, deploy, undeploy, delete, export, check existence in the Metadata Store (MDS). |
| MFT Metadata Commands | Export, import, and reset MFT metadata. |

*Table 5-1 (Cont.) MFT WLST Command Categories*

| Command Category | Description |
|---|---|
| MFT Key Management Commands | Generate, import, export, delete, list, and update SSL, SSH, and PGP keys in the MFT keystore. |
| MFT Deployment History Commands | View the deployment history of source, transfer, and target artifacts. |
| MFT Transfer Management Commands | Pause, resume, resubmit, or display information about transfer instances. |
| MFT Embedded Server Commands | Start, stop, and change ports of embedded FTP and sFTP servers. |
| MFT Callout Commands | Create, delete, list, and update custom callouts. |
| MFT Event Notification Commands | • Create, delete, and list contacts<br>• Add and remove contacts from event notifications<br>• Enable and disable event notifications |
| MFT Archive and Restore Commands | Archive transfer instances or file system data based on criteria such as status and date ranges. |
| MFT Purge Commands | Purge transfer instances or file system data based on criteria such as status and date ranges. |

# MFT Artifact Management Commands

Use the MFT WLST Artifact Management commands, listed in Table 5-2, to perform operations on source, transfer, and target artifacts.

*Table 5-2 MFT Artifact Management WLST Commands*

| Use this command... | To... | Use with WLST |
|---|---|---|
| bulkDeployArtifact | Deploys multiple source, transfer, or target artifacts. | Online or Offline |
| deleteArtifact or `delAF` | Delete a source, transfer, or target artifact. | Online or Offline |
| deleteArtifactDeployment or `delDepAF` | Delete an undeployed source, transfer, or target artifact. | Online or Offline |
| deployArtifact or `depAF` | Deploy a source, transfer, or target artifact. | Online or Offline |
| disableArtifact or `disAF` | Disable a deployed source, transfer, or target artifact. | Online or Offline |
| enableArtifact or `enAF` | Enable a deployed source, transfer, or target artifact. | Online or Offline |
| exportDeployedArtifact or `expDepAF` | Export a deployed source, transfer, or target artifact to a ZIP file. | Online or Offline |
| isArtifactInMDS or `isAFinMDS` | Check whether a source, transfer, or target artifact exists in the MDS (Metadata Store). | Online or Offline |

*Table 5-2 (Cont.) MFT Artifact Management WLST Commands*

| Use this command... | To... | Use with WLST |
|---|---|---|
| undeployArtifact or undepAF | Undeploy a source, transfer, or target artifact. | Online or Offline |

## bulkDeployArtifact

Command Category: MFT Artifact Management Commands

Use with WLST: Online or Offline

### Description

Bulk deploys multiple source, transfer, or target artifacts. In the event of an unsupported operation, the command returns a `WLSTException`.

### Syntax

```
bulkDeployArtifact(artifact_type, artifact_names, comment)
```

| Argument | Definition |
|---|---|
| *artifact_type* | Artifact type: SOURCE, TRANSFER, or TARGET. |
| *artifact_names* | Comma-separated artifact names, or * for all. |
| *comment* | Text string describing the artifacts or the reason for deploying them. |

### Example

The following example deploys two SOURCE artifacts and provides a comment.

```
bulkDeployArtifact('SOURCE', 'order-file-src,chk-inv-src', 'retrieves new purchase
orders, checks inventory')
```

The following example deploys all TRANSFER artifacts and provides a comment.

```
bulkDeployArtifact('TRANSFER', '*', 'deploying all transfers')
```

## deleteArtifact

Command Category: MFT Artifact Management Commands

Use with WLST: Online or Offline

### Description

Deletes a source, transfer, or target artifact. In the event of an unsupported operation, the command returns a `WLSTException`.

### Syntax

```
deleteArtifact(artifact_type, artifact_name)
```

```
delAF(artifact_type, artifact_name)
```

| Argument | Definition |
|---|---|
| *artifact_type* | Artifact type: SOURCE, TRANSFER, or TARGET. |
| *artifact_name* | Artifact name. |

### Example

The following example deletes a SOURCE artifact called `order-file-src`.

```
deleteArtifact('SOURCE', 'order-file-src')
```

## deleteArtifactDeployment

Command Category: MFT Artifact Management Commands

Use with WLST: Online or Offline

### Description

Deletes an undeployed source, transfer, or target artifact. In the event of an unsupported operation, the command returns a `WLSTException`.

### Syntax

```
deleteArtifactDeployment(artifact_type, artifact_name, label)
```

```
delDepAF(artifact_type, artifact_name, label)
```

| Argument | Definition |
|---|---|
| *artifact_type* | Artifact type: SOURCE, TRANSFER, or TARGET. |
| *artifact_name* | Artifact name. |
| *label* | Artifact label in the MDS (Metadata Store).<br>Use **Show Deployment Details** on the Deployment tab to view this label. |

### Example

The following example deletes a SOURCE artifact called `order-file-src` with the label `soa_mft-2012-12-07 22:24:09.383`.

```
deleteArtifactDeployment('SOURCE', 'order-file-src', 'soa_mft-2012-12-07
22:24:09.383')
```

## deployArtifact

Command Category: MFT Artifact Management Commands

Use with WLST: Online or Offline

### Description

Deploys a source, transfer, or target artifact. In the event of an unsupported operation, the command returns a `WLSTException`.

### Syntax

`deployArtifact(`*`artifact_type`*`, `*`artifact_name`*`, `*`comment`*`)`

`depAF(`*`artifact_type`*`, `*`artifact_name`*`, `*`comment`*`)`

| Argument | Definition |
|---|---|
| *artifact_type* | Artifact type: `SOURCE`, `TRANSFER`, or `TARGET`. |
| *artifact_name* | Artifact name. |
| *comment* | Text string describing the artifact or the reason for deploying it. |

### Example

The following example deploys a `SOURCE` artifact called `order-file-src` and provides a comment about the artifact.

```
deployArtifact('SOURCE', 'order-file-src', 'retrieves new purchase orders')
```

## disableArtifact

Command Category: MFT Artifact Management Commands

Use with WLST: Online or Offline

### Description

Disables a deployed and previously enabled source, transfer, or target artifact. In the event of an unsupported operation, the command returns a `WLSTException`.

### Syntax

`disableArtifact(`*`artifact_type`*`, `*`artifact_name`*`, `*`comment`*`)`

`disAF(`*`artifact_type`*`, `*`artifact_name`*`, `*`comment`*`)`

| Argument | Definition |
|---|---|
| *artifact_type* | Artifact type: `SOURCE`, `TRANSFER`, or `TARGET`. |
| *artifact_name* | Artifact name. |
| *comment* | Text string describing the artifact or the reason for disabling it. |

### Example

The following example disables a SOURCE artifact called order-file-src and provides a comment about the reason.

```
disableArtifact('SOURCE', 'order-file-src', 'order server being upgraded')
```

## enableArtifact

Command Category: MFT Artifact Management Commands

Use with WLST: Online or Offline

### Description

Enables a deployed and previously disabled source, transfer, or target artifact. In the event of an unsupported operation, the command returns a WLSTException.

### Syntax

```
enableArtifact(artifact_type, artifact_name, comment)

enAF(artifact_type, artifact_name, comment)
```

| Argument | Definition |
|---|---|
| *artifact_type* | Artifact type: SOURCE, TRANSFER, or TARGET. |
| *artifact_name* | Artifact name. |
| *comment* | Text string describing the artifact or the reason for enabling it. |

### Example

The following example enables a SOURCE artifact called order-file-src and provides a comment about the reason.

```
enableArtifact('SOURCE', 'order-file-src', 'order server upgrade complete')
```

## exportDeployedArtifact

Command Category: MFT Artifact Management Commands

Use with WLST: Online or Offline

### Description

Exports a deployed source, transfer, or target artifact to a ZIP file. In the event of an unsupported operation, the command returns a WLSTException.

### Syntax

```
exportDeployedArtifact(artifact_type, artifact_name, label, archive_file_path)

expDepAF(artifact_type, artifact_name, label, archive_file_path)
```

| Argument | Definition |
|---|---|
| *artifact_type* | Artifact type: SOURCE, TRANSFER, or TARGET. |
| *artifact_name* | Artifact name. |
| *label* | Artifact label in the MDS (Metadata Store).<br>Use **Show Deployment Details** on the Deployment tab to view this label. |
| *archive_file_path* | Full path to the ZIP file to which to export. If you are connecting to WLST remotely, the ZIP file is created on the remote server. |

### Example

The following example exports a SOURCE artifact called `order-file-src` with the label `soa_mft-2012-12-07 22:24:09.383` to `/export/order-file-src.zip`.

```
exportDeployedArtifact('SOURCE', 'order-file-src', 'soa_mft-2012-12-07
22:24:09.383', '/export/order-file-src.zip')
```

## isArtifactInMDS

Command Category: MFT Artifact Management Commands

Use with WLST: Online or Offline

### Description

Checks whether a source, transfer, or target artifact exists in the MDS (Metadata Store) and returns TRUE or FALSE. In the event of an unsupported operation, the command returns a `WLSTException`.

### Syntax

```
isArtifactInMDS(artifact_type, artifact_name)
```

```
isAFinMDS(artifact_type, artifact_name)
```

| Argument | Definition |
|---|---|
| *artifact_type* | Artifact type: SOURCE, TRANSFER, or TARGET. |
| *artifact_name* | Artifact name. |

### Example

The following example checks whether a SOURCE artifact called `order-file-src` exists in the MDS.

```
isArtifactInMDS('SOURCE', 'order-file-src')
```

## undeployArtifact

Command Category: MFT Artifact Management Commands

Use with WLST: Online or Offline

### Description

Undeploys a source, transfer, or target artifact without deleting it from the configuration. In the event of an unsupported operation, the command returns a `WLSTException`.

### Syntax

```
undeployArtifact(artifact_type, artifact_name, comment)

undepAF(artifact_type, artifact_name, comment)
```

| Argument | Definition |
|---|---|
| *artifact_type* | Artifact type: `SOURCE`, `TRANSFER`, or `TARGET`. |
| *artifact_name* | Artifact name. |
| *comment* | Text string describing the artifact or the reason for undeploying it. |

### Example

The following example undeploys a `SOURCE` artifact called `order-file-src` and provides a comment about the reason.

```
undeployArtifact('SOURCE', 'order-file-src', 'upgrading artifact')
```

# MFT Metadata Commands

Use the MFT WLST Metadata commands, listed in Table 5-3, to perform metadata operations.

***Table 5-3 MFT Metadata WLST Commands***

| Use this command... | To... | Use with WLST |
|---|---|---|
| exportMetadata or `expMD` | Export the entire MFT configuration, excluding passwords, to a ZIP file. | Online or Offline |
| exportTransferMetadata or `expXfrMD` | Export a transfer artifact and related metadata to a ZIP file. | Online or Offline |
| importMetadata or `impMD` | Import a previously exported MFT configuration from a ZIP file. | Online or Offline |

*Table 5-3 (Cont.) MFT Metadata WLST Commands*

| Use this command... | To... | Use with WLST |
|---|---|---|
| resetMetadata or `resMD` | Reset the MFT configuration, deleting all artifacts and resetting all administrative settings to their defaults, while optionally preserving user preferences. | Online or Offline |

## exportMftMetadata

Command Category: MFT Metadata Commands

Use with WLST: Online or Offline

### Description

Exports the entire MFT configuration, excluding passwords, to a ZIP file. In the event of an unsupported operation, the command returns a `WLSTException`.

### Syntax

```
exportMftMetadata(archive_file_path)

expMD(archive_file_path)
```

| Argument | Definition |
|---|---|
| *archive_file_path* | Full path to the ZIP file to which to export. |

### Example

The following example exports the MFT configuration to `/export/mft-config-dec-2012.zip`.

```
exportMftMetadata('/export/mft-config-dec-2012.zip')
```

## exportTransferMetadata

Command Category: MFT Metadata Commands

Use with WLST: Online or Offline

### Description

Exports a transfer artifact and related metadata to a ZIP file. In the event of an unsupported operation, the command returns a `WLSTException`.

### Syntax

```
exportTransferMetadata(archive_file_path, transfer_name)

expXfrMD(archive_file_path, transfer_name)
```

| Argument | Definition |
|---|---|
| *archive_file_path* | Full path to the ZIP file to which to export. |
| *transfer_name* | Transfer artifact name. |

### Example

The following example exports a transfer artifact named `order-xfr` and its metadata to `/export/order-xfr.zip`.

```
exportTransferMetadata('/export/order-xfr.zip', 'order-xfr')
```

## importMftMetadata

Command Category: MFT Metadata Commands

Use with WLST: Online or Offline

### Description

Imports a previously exported MFT configuration from a ZIP file. In the event of an unsupported operation, the command returns a `WLSTException`.

### Syntax

```
importMftMetadata(archive_file_path)

impMD(archive_file_path)
```

| Argument | Definition |
|---|---|
| *archive_file_path* | Full path to the ZIP file to which to export. |

### Example

The following example imports the MFT configuration from `/export/mft-config-dec-2012.zip`.

```
importMftMetadata('/export/mft-config-dec-2012.zip')
```

## resetMetadata

Command Category: MFT Metadata Commands

Use with WLST: Online or Offline

### Description

Resets the MFT configuration, deleting all artifacts and resetting all administrative settings to their defaults, while optionally preserving user preferences. In the event of an unsupported operation, the command returns a `WLSTException`.

### Syntax

```
resetMetadata(preserve_preferences)

resMD(preserve_preferences)
```

| Argument | Definition |
|---|---|
| *preserve_preferences* | If TRUE, preserves user preferences. If FALSE, resets user preferences along with the rest of the configuration. |

### Example

The following example resets the MFT configuration but preserves user preferences.

```
resetMetadata('TRUE')
```

## MFT Key Management Commands

Use the MFT WLST Key Management commands, listed in Table 5-4, to manage SSL, SSH, and PGP keys in the MFT keystore. For additional information, see "Keystore Management" in *Using Oracle Managed File Transfer*.

***Table 5-4 MFT Key Management WLST Commands***

| Use this command... | To... | Use with WLST |
|---|---|---|
| deleteCSFKey or `delKey` | Delete a key alias from the MFT keystore. | Online or Offline |
| exportCSFKey or `expKey` | Export keys from the MFT keystore to a key file. | Online or Offline |
| generateKeys or `genKeys` | Generate keys and save them to one or more key files. | Online or Offline |
| importCSFKey or `impKey` | Import a key to the MFT keystore from a key file. | Online or Offline |
| listCSFKeyAliases or `lsKeyAliases` | List key aliases in the MFT keystore. | Online or Offline |
| updateCSFKey or `updKey` | Delete a key alias from the MFT keystore and generate a new key file. | Online or Offline |

## deleteCSFKey

Command Category: MFT Key Management Commands

Use with WLST: Online or Offline

### Description

Deletes a key alias from the MFT keystore. In the event of an unsupported operation, the command returns a `WLSTException`.

### Syntax

```
deleteCSFKey(key_format, key_type, alias)
```

```
delKey(key_format, key_type, alias)
```

| Argument | Definition |
|---|---|
| *key_format* | Key format: SSH or PGP. |
| *key_type* | Key format: PRIVATE or PUBLIC. |
| *alias* | Key alias. |

### Example

The following example deletes a PRIVATE SSH key with the alias my-alias.

```
deleteCSFKey('SSH', 'PRIVATE', 'my-alias')
```

## exportCSFKey

Command Category: MFT Key Management Commands

Use with WLST: Online or Offline

### Description

Exports keys from the MFT keystore to a key file. In the event of an unsupported operation, the command returns a WLSTException.

### Syntax

```
exportCSFKey(key_format, key_type, key_file_path)
```

```
expKey(key_format, key_type, key_file_path)
```

| Argument | Definition |
|---|---|
| *key_format* | Key format: SSH or PGP. |
| *key_type* | Key format: PRIVATE or PUBLIC. |
| *key_file_path* | Full path to the key file to which to export. |

### Example

The following example exports PRIVATE SSH keys to the file /export/ssh/my_private_keys.ppk.

```
exportCSFKey('SSH', 'PRIVATE', '/export/ssh/my_private_keys.ppk')
```

## generateKeys

Command Category: MFT Key Management Commands

Use with WLST: Online or Offline

### Description

Generates keys and saves them to one or more key files. The key type is RSA and the key size is 1024 bits. In the event of an unsupported operation, the command returns a `WLSTException`.

### Syntax

```
generateKeys(key_format, password, key_file_path)

genKeys(key_format, password, key_file_path)
```

| Argument | Definition |
|---|---|
| *key_format* | Key format: SSH or PGP. |
| *password* | Optional password for the private key. |
| *key_file_path* | Full path to the generated key file or directory.<br>For SSH, the path must include the key file name.<br>For PGP, two files are generated under the specified path: the `secret.asc` file contains the PGP private key, and the `pub.asc` file contains the PGP public key. |

### Example

The following example exports SSH keys without password protection to the file `/export/ssh/ssh-pvt-keys.ppk`.

```
generateKeys('SSH', '','/export/ssh/ssh-pvt-keys.ppk')
```

The following example exports PGP keys with password protection to the directory `/export/pgp`.

```
generateKeys('PGP', 'P@s$W0rd','/export/pgp')
```

## importCSFKey

Command Category: MFT Key Management Commands

Use with WLST: Online or Offline

### Description

Imports a key to the MFT keystore from a key file and creates an alias. In the event of an unsupported operation, the command returns a `WLSTException`.

### Syntax

```
importCSFKey(key_format, key_type, alias, key_file_path)

impKey(key_format, key_type, alias, key_file_path)
```

| Argument | Definition |
|---|---|
| *key_format* | Key format: SSH or PGP. |
| *key_type* | Key format: PRIVATE or PUBLIC. |
| *alias* | Key alias. |
| *key_file_path* | Full path to the key file from which to import. |

### Example

The following example imports a PRIVATE SSH key with the alias my-alias from the file /export/ssh/my_private_keys.ppk.

```
importCSFKey('SSH', 'PRIVATE', 'my-alias', '/export/ssh/my_private_keys.ppk')
```

The following example imports a PUBLIC PGP key with the alias mftpgppub from the file /export/pgp/pub.asc.

```
importCSFKey('PGP', 'PUBLIC', 'mftpgppub', '/export/pgp/pub.asc')
```

The following example imports a PRIVATE PGP key with the alias mftpgppri from the file /export/pgp/secret.asc.

```
importCSFKey('PGP', 'PRIVATE', 'mftpgppri', '/export/pgp/secret.asc')
```

## listCSFKeyAliases

Command Category: MFT Key Management Commands

Use with WLST: Online or Offline

### Description

Lists key aliases in the MFT keystore. In the event of an unsupported operation, the command returns a WLSTException.

### Syntax

```
listCSFKeyAliases(key_format, key_type, alias)
```

```
lsKeyAliases(key_format, key_type, alias)
```

| Argument | Definition |
|---|---|
| *key_format* | Key format: SSH or PGP. |
| *key_type* | Key format: PRIVATE or PUBLIC. |

### Example

The following example lists PRIVATE SSH keys.

```
listCSFKeyAliases('SSH', 'PRIVATE')
```

## updateCSFKey

Command Category: MFT Key Management Commands

Use with WLST: Online or Offline

### Description

Deletes a key alias from the MFT keystore and generates a new key file. In the event of an unsupported operation, the command returns a WLSTException.

### Syntax

```
updateCSFKey(key_format, key_type, alias, key_file_path)
```

```
updKey(key_format, key_type, alias, key_file_path)
```

| Argument | Definition |
|---|---|
| *key_format* | Key format: SSH or PGP. |
| *key_type* | Key format: PRIVATE or PUBLIC. |
| *alias* | Key alias. |
| *key_file_path* | Full path to the key file to generate. |

### Example

The following example deletes a PRIVATE SSH key with the alias my-alias and generates the key file /export/ssh/my-private-key.ppk.

```
updateCSFKey('SSH', 'PRIVATE', 'my-alias', '/export/ssh/my-private-key.ppk')
```

# MFT Deployment History Commands

Use the MFT WLST Deployment History commands, listed in Table 5-5, to view the deployment history of source, transfer, and target artifacts.

***Table 5-5 MFT Deployment History WLST Commands***

| Use this command... | To... | Use with WLST |
|---|---|---|
| getSourceDeploymentHistory or getSrcDH | Return the deployment history of a source artifact. | Online or Offline |

*Table 5-5 (Cont.) MFT Deployment History WLST Commands*

| Use this command... | To... | Use with WLST |
|---|---|---|
| getTargetDeploymentHistory or `getTrgtDH` | Return the deployment history of a target artifact. | Online or Offline |
| getTransferDeploymentHistory or `getXfrDH` | Return the deployment history of a transfer artifact. | Online or Offline |

## getSourceDeploymentHistory

Command Category: MFT Deployment History Commands

Use with WLST: Online or Offline

### Description

Returns the deployment history of a source artifact. In the event of an unsupported operation, the command returns a `WLSTException`.

### Syntax

```
getSourceDeploymentHistory(source_name)

getSrcDH(source_name)
```

| Argument | Definition |
|---|---|
| *source_name* | Source artifact name. |

### Example

The following example returns the deployment history of a source artifact named `order-file-src`.

```
getSourceDeploymentHistory('order-file-src')
```

## getTargetDeploymentHistory

Command Category: MFT Deployment History Commands

Use with WLST: Online or Offline

### Description

Returns the deployment history of a target artifact. In the event of an unsupported operation, the command returns a `WLSTException`.

### Syntax

```
getTargetDeploymentHistory(target_name)

getTrgtDH(target_name)
```

| Argument | Definition |
|---|---|
| *target_name* | Target artifact name. |

### Example

The following example returns the deployment history of a target artifact named `order-file-tgt`.

```
getTargetDeploymentHistory('order-file-tgt')
```

## getTransferDeploymentHistory

Command Category: MFT Deployment History Commands

Use with WLST: Online or Offline

### Description

Returns the deployment history of a transfer artifact. In the event of an unsupported operation, the command returns a `WLSTException`.

### Syntax

```
getTransferDeploymentHistory(transfer_name)

getXfrDH(transfer_name)
```

| Argument | Definition |
|---|---|
| *transfer_name* | Transfer artifact name. |

### Example

The following example returns the deployment history of a transfer artifact named `order-xfr`.

```
getTransferDeploymentHistory('order-xfr')
```

# MFT Transfer Management Commands

Use the MFT WLST Transfer Management commands, listed in Table 5-6, to manage transfer instances.

***Table 5-6 MFT Transfer Management WLST Commands***

| Use this command... | To... | Use with WLST |
|---|---|---|
| getTransferInfo or `getXfrInfo` | Return information about a transfer artifact. | Online or Offline |
| pauseTransfer or `pauseXfr` | Pause an in-progress transfer. | Online or Offline |
| resubmit or `resub` | Resubmit a failed transfer. | Online or Offline |

*Table 5-6 (Cont.) MFT Transfer Management WLST Commands*

| Use this command... | To... | Use with WLST |
|---|---|---|
| resumeTransfer or `resXfr` | Resume a paused transfer. | Online or Offline |

## getTransferInfo

Command Category: MFT Transfer Management Commands

Use with WLST: Online or Offline

### Description

Returns information about a transfer artifact. In the event of an unsupported operation, the command returns a `WLSTException`.

### Syntax

```
getTransferInfo(transfer_name, label)

getXfrInfo(transfer_name, label)
```

| Argument | Definition |
|---|---|
| *transfer_name* | Transfer artifact name. |
| *label* | Artifact label in the MDS (Metadata Store).<br>Use **Show Deployment Details** on the Deployment tab to view this label. |

### Example

The following example returns information about a transfer artifact named `order-xfr` with the label `soa_mft-2012-12-07 22:28:17.392`.

```
getTransferInfo('order-xfr', 'soa_mft-2013-09-26 22:28:17.392')
ID                                       | NAME          | LABEL
xfer_521a6788-4e24-4822-bb03-43a4a8eaa8ce | ftp-file-xfer | soa_mft-2013-09-26
16:00:09.803
```

## pauseTransfer

Command Category: MFT Transfer Management Commands

Use with WLST: Online or Offline

### Description

Pauses an in-progress transfer. In the event of an unsupported operation, the command returns a `WLSTException`.

### Syntax

```
pauseTransfer(instance_id, comment)
```

```
pauseXfr(instance_id, comment)
```

| Argument | Definition |
|---|---|
| *instance_id* | Target instance ID for the transfer.<br>Open the Advanced section of the target report to view the instance ID. For information about the target report, see "Interpreting Source, Transfer, and Target Reports" in *Using Oracle Managed File Transfer*. |
| *comment* | Text string describing the reason for pausing. |

### Example

The following example pauses a transfer with the target instance ID `240C93AD-5401-483B-8182-274FA0705DF1` and provides a reason.

```
pauseTransfer('240C93AD-5401-483B-8182-274FA0705DF1', 'resume when less network
traffic')
```

## resubmit

Command Category: MFT Transfer Management Commands

Use with WLST: Online or Offline

### Description

Resubmits a transfer. You can resubmit a successful or failed transfer. In the event of an unsupported operation, the command returns a `WLSTException`.

### Syntax

```
resubmit(resubmit_type, instance_ids, comment)

resub(resubmit_type, instance_ids, comment)
```

| Argument | Definition |
|---|---|
| *resubmit_type* | Resubmit type: `SOURCE`, `TRANSFER_INSTANCE`, `TARGET`, or `TARGET_INSTANCE`.<br>For `TARGET`, the file is delivered without preprocessing. For `TARGET_INSTANCE`, any preprocessing is called again before delivery. |
| *instance_ids* | Comma-separated instance IDs.<br>Open the Advanced section of the source, transfer, or target report to view the instance ID. For information about these reports, see "Interpreting Source, Transfer, and Target Reports" in *Using Oracle Managed File Transfer*. |
| *comment* | Text string describing the reason for resubmitting. |

### Example

The following example resubmits a `SOURCE` with the instance ID `3D48B12B-295A-4F52-A8EE-BD1CC1A20246` and provides a reason.

```
resubmit('SOURCE', '3D48B12B-295A-4F52-A8EE-BD1CC1A20246', 'trying again')
```

## resumeTransfer

Command Category: MFT Transfer Management Commands

Use with WLST: Online or Offline

### Description

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

### Syntax

```
resumeTransfer(instance_id, comment)

resXfr(instance_id, comment)
```

| Argument | Definition |
|---|---|
| *instance_id* | Target instance ID for the transfer.<br>Open the Advanced section of the target report to view the instance ID. For information about the target report, see "Interpreting Source, Transfer, and Target Reports" in *Using Oracle Managed File Transfer*. |
| *comment* | Text string describing the reason for resuming. |

### Example

The following example resumes a transfer with the target instance ID `240C93AD-5401-483B-8182-274FA0705DF1` and provides a reason.

```
resumeTransfer('240C93AD-5401-483B-8182-274FA0705DF1', 'less network traffic now')
```

# MFT Embedded Server Commands

Use the MFT WLST Embedded Server commands, listed in Table 5-7, to manage embedded FTP and sFTP servers.

***Table 5-7 MFT Embedded Server WLST Commands***

| Use this command... | To... | Use with WLST |
|---|---|---|
| ConfigureHomeDir or `confHmDir` | Assigns a home directory on an embedded server. | Online or Offline |
| grantPermissionToDirectory or `grPermDir` | Grant permission to an embedded server directory. | Online or Offline |

*Table 5-7 (Cont.) MFT Embedded Server WLST Commands*

| Use this command... | To... | Use with WLST |
|---|---|---|
| listAllPermissions or `lsPerms` | List all permissions available for a given principal and server type. | Online or Offline |
| revokePermissionForDirectory or `revPermDir` | Revoke permissions of an embedded server directory. | Online or Offline |
| startEmbeddedServer or `startES` | Start an embedded FTP or sFTP server that was stopped. | Online or Offline |
| stopEmbeddedServer or `stopES` | Stop an embedded FTP or sFTP server that is running. | Online or Offline |
| updatePorts or `updPorts` | Update the port for an embedded FTP or sFTP server. | Online or Offline |

## ConfigureHomeDir

Command Category: MFT Embedded Server Commands

Use with WLST: Online or Offline

### Description

Assigns the specified directory to the user as home directory where that user is located on login to embedded servers.

### Syntax

`configureHomeDir(`*`directory_path, user_name`*`)`

| Argument | Definition |
|---|---|
| `directory_path` | Any valid path starting from the embedded server root. |
| `user_name` | A WebLogic user ID |

### Example

The following example assigns directory `dir1` as the home directory for a user named "user1".

```
configureHomeDir("/dir1","user1")
```

## grantPermissionToDirectory

Command Category: MFT Embedded Server Commands

Use with WLST: Online or Offline

### Description

Grant permission to an embedded server directory. Users and groups can be assigned a set of permissions to an existing directory on an embedded server.

### Syntax

```
grantPermissionToDirectory(directory_path, principal_name, principal_type,
permissions, server_type, include_subfolder)
```

| Argument | Definition |
|---|---|
| directory_path | Any valid path starting from the embedded server root |
| principal_name | An enterprise user, group or application role name. |
| principal_type | One of the following:<br>• USER<br>• GROUP<br>• APPLICATION_ROLE |
| permissions | Comma separated list of the following permissions:<br>• READ<br>• WRITE<br>• LIST<br>• DELETE |
| server_type | FTP or SFTP |
| include_subfolder | true or false |

### Example

The following example assigns read, write, list and delete permissions to the directory /orders (and sub-directories) for the user "weblogic" on an FTP server:

```
grantPermissionToDirectory("/orders","weblogic","USER", "READ, WRITE, LIST,
DELETE","FTP",true)
```

## listAllPermissions

Command Category: MFT Embedded Server Commands

Use with WLST: Online or Offline

### Description

List all permissions available for a given principal and server type. The server type can be FTP or SFTP.

### Syntax

```
listAllPermissions(principal_name,server_type)
```

| Argument | Definition |
|---|---|
| principal_name | An enterprise user, group or application role name. |
| server_type | FTP or SFTP |

### Example

The following example lists all permissions for the principal named "weblogic" on an FTP server.

```
listAllPermissions("weblogic","FTP"
```

## revokePermissionForDirectory

Command Category: MFT Embedded Server Commands

Use with WLST: Online or Offline

### Description

Revoke a set of permissions from an embedded server directory.

### Syntax

`revokePermissionForDirectory(`*`directory_path, principal_name, principal_type, permissions, server_type, include_subfolder`*`)`

| Argument | Definition |
|---|---|
| `directory_path` | Any valid path starting from the embedded server root |
| `principal_name` | An enterprise user, group or application role name. |
| `principal_type` | One of the following:<br>• `USER`<br>• `GROUP`<br>• `APPLICATION_ROLE` |
| `permissions` | Comma separated list of the following permissions:<br>• `READ`<br>• `WRITE`<br>• `LIST`<br>• `DELETE` |
| `server_type` | `FTP` or `SFTP` |
| `include_subfolder` | `true` or `false` |

### Example

The following example revokes read, write, list and delete permissions from the directory `/orders` (and sub-directories) for the user "weblogic" on an FTP server:

```
grantPermissionToDirectory("/orders","weblogic","USER", "READ, WRITE, LIST,
DELETE","FTP",true)
```

## startEmbeddedServer

Command Category: MFT Embedded Server Commands

Use with WLST: Online or Offline

### Description

Starts an embedded FTP, FTPS (FTP over SSL), or sFTP (SSH-FTP) server that was stopped. In the event of an unsupported operation, the command returns a `WLSTException`.

### Syntax

```
startEmbeddedServer(server_type)

startES(server_type)
```

| Argument | Definition |
|---|---|
| *server_type* | Embedded server type: FTP, FTPS, or SFTP. |

### Example

The following example starts the embedded FTP server.

```
startEmbeddedServer('FTP')
```

## stopEmbeddedServer

Command Category: MFT Embedded Server Commands

Use with WLST: Online or Offline

### Description

Stops an embedded FTP, FTPS (FTP over SSL), or sFTP (SSH-FTP) server that is running. In the event of an unsupported operation, the command returns a `WLSTException`.

### Syntax

```
stopEmbeddedServer(server_type)

stopES(server_type)
```

| Argument | Definition |
|---|---|
| *server_type* | Embedded server type: FTP, FTPS, or SFTP. |

### Example

The following example stops the embedded FTP server.

```
stopEmbeddedServer('FTP')
```

## updatePorts

Command Category: MFT Embedded Server Commands

Use with WLST: Online or Offline

### Description

Updates the port for an embedded FTP, FTPS (FTP over SSL), or sFTP (SSH-FTP) server, which is a service of an Oracle WebLogic Server managed server dedicated to MFT. In the event of an unsupported operation, the command returns a `WLSTException`.

### Syntax

```
updatePorts(server_instance, service, port)

updPorts(server_instance, service, port)
```

| Argument | Definition |
|---|---|
| *server_instance* | Name of Oracle WebLogic Server managed server dedicated to MFT. |
| *service* | Service (embedded server type): `FTP`, `FTPS`, or `SFTP`. |
| *port* | Port number. |

### Example

The following example updates the port of the embedded `FTP` server.

```
updatePorts('mft_server1', 'FTP', 7021)
```

## MFT Callout Commands

Use the MFT WLST Callout commands, listed in Table 5-8, to manage custom callouts.

***Table 5-8 MFT Callout WLST Commands***

| Use this command... | To... | Use with WLST |
|---|---|---|
| createCallouts or `crtCalls` | Create callouts based on an XML file that defines them. | Online or Offline |
| deleteCallout or `delCalls` | Delete a callout. | Online or Offline |
| listCallouts or `lsCalls` | List callouts. | Online or Offline |
| updateCallouts or `updCalls` | Update callouts with the same names based on an XML file that defines them. | Online or Offline |

## createCallouts

Command Category: MFT Callout Commands

Use with WLST: Online or Offline

### Description

Creates callouts based on an XML file that defines them. In the event of an unsupported operation, the command returns a `WLSTException`.

See Processing Transfers with Custom Callouts in *Using Oracle Managed File Transfer* for callout instructions and examples.

### Syntax

```
createCallouts(def_file_path)

crtCalls(def_file_path)
```

| Argument | Definition |
|---|---|
| *def_file_path* | Full path to the callout definition XML file. |

### Example

The following example creates callouts based on a definition file named `/tmp/CalloutDefn.xml`.

```
createCallouts('/tmp/CalloutDefn.xml')
```

## deleteCallout

Command Category: MFT Callout Commands

Use with WLST: Online or Offline

### Description

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

### Syntax

```
deleteCallout(callout_name)

delCalls(callout_name)
```

| Argument | Definition |
|---|---|
| *callout_name* | Callout name. |

### Example

The following example deletes a callout named `NotifyTransferComplete`.

```
deleteCallout('NotifyTransferComplete')
```

## listCallouts

Command Category: MFT Callout Commands

Use with WLST: Online or Offline

### Description

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

### Syntax

```
listCallouts()

lsCalls()
```

### Example

The following example lists callouts.

```
listCallouts()
```

## updateCallouts

Command Category: MFT Callout Commands

Use with WLST: Online or Offline

### Description

Updates callouts with the same names based on an XML file that defines them. In the event of an unsupported operation, the command returns a `WLSTException`.

**Note:**

Parameters of existing callouts cannot be added, deleted, or modified.

### Syntax

```
updateCallouts(def_file_path)

updCalls(def_file_path)
```

| Argument | Definition |
|---|---|
| *def_file_path* | Full path to the callout definition XML file. |

### Example

The following example updates callouts based on a definition file named `/tmp/CalloutDefn.xml`.

```
updateCallouts('/tmp/CalloutDefn.xml')
```

# MFT Event Notification Commands

Use the MFT WLST Event Notification commands, listed in Table 5-9, to manage contact notifications of events.

*Table 5-9 MFT Contact WLST Commands*

| Use this command... | To... | Use with WLST |
|---|---|---|
| addContactToNotification or `addContNote` | Add a contact to a specific event notification. | Online or Offline |
| createContact or `CrtCont` | Create a contact for event notifications. | Online or Offline |
| deleteContact or `delCont` | Delete a contact. | Online or Offline |
| listContacts or `lsConts` | List contacts. | Online or Offline |
| removeContactFromNotification or `remContNote` | Remove a contact from a specific event notification. | Online or Offline |
| updateEvent or `updEvt` | Enable or disable a specific event notification. | Online or Offline |

## addContactToNotification

Command Category: MFT Contact Commands

Use with WLST: Online or Offline

### Description

Adds a contact to a specific event notification. In the event of an unsupported operation, the command returns a `WLSTException`.

### Syntax

```
addContactToNotification(event, contact_type, value)

addContNote(event, contact_type, value)
```

| Argument | Definition |
|---|---|
| *event* | Event: `RUNTIME_ERROR_EVENT`, `DELETE_ARTIFACT_EVENT`, `DEPLOY_ARTIFACT_EVENT`, `EXPORT_IMPORT_EVENT`, `PURGE_EVENT`, or `ARCHIVE_RESTORE_EVENT`. |
| *contact_type* | Contact type: `EMAIL`, `PHONE`, `FAX`, or `SMS`. |
| *value* | Email address or phone number. |

### Example

The following example adds `EMAIL` contact `jane.doe@example.com` to `RUNTIME_ERROR_EVENT` notifications.

```
addContactToNotification('RUNTIME_ERROR_EVENT', 'EMAIL', 'jane.doe@example.com')
```

## createContact

Command Category: MFT Contact Commands

Use with WLST: Online or Offline

### Description

Creates a contact for event notifications. In the event of an unsupported operation, the command returns a `WLSTException`.

See Configuring an Email Driver for Notifications in *Using Oracle Managed File Transfer* for information on how to configure an email driver. See Configuring an SMS Driver for Notifications in *Using Oracle Managed File Transfer* for information on how to configure an SMS driver.

**Note:**

Phone and FAX notifications are not supported in this release of Oracle Managed File Transfer.

### Syntax

```
createContact(contact_type, value)

crtCont(contact_type, value)
```

| Argument | Definition |
|---|---|
| *contact_type* | Contact type: `EMAIL`, `PHONE`, `FAX`, or `SMS`. |
| *value* | Email address or phone number. |

### Example

The following example creates a contact based on the email address `jane.doe@example.com`.

```
createContact('EMAIL', 'jane.doe@example.com')
```

## deleteContact

Command Category: MFT Contact Commands

Use with WLST: Online or Offline

### Description

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

### Syntax

```
deleteContact(contact_type, value)
```

```
delCont(contact_type, value)
```

| Argument | Definition |
|---|---|
| *contact_type* | Contact type: `EMAIL`, `PHONE`, `FAX`, or `SMS`. |
| *value* | Email address or phone number. |

### Example

The following example deletes a contact based on the email address `jane.doe@example.com`.

```
deleteContact('EMAIL', 'jane.doe@example.com')
```

## listContacts

Command Category: MFT Contact Commands

Use with WLST: Online or Offline

### Description

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

### Syntax

```
listContacts(contact_type)
```

```
lsConts(contact_type)
```

| Argument | Definition |
|---|---|
| *contact_type* | Contact type: `EMAIL`, `PHONE`, `FAX`, or `SMS`. |

### Example

The following example lists email contacts.

```
listContacts('EMAIL')
```

## removeContactFromNotification

Command Category: MFT Contact Commands

Use with WLST: Online or Offline

### Description

Removes a contact from a specific event notification. In the event of an unsupported operation, the command returns a `WLSTException`.

When you remove the last contact from an event, the event is disabled. However, the reverse is not true. When you add the first contact to an event, it is not enabled automatically. Enable an event using the updateEvent command.

### Syntax

```
removeContactFromNotification(event, contact_type, value)

remContNote(event, contact_type, value)
```

| Argument | Definition |
|---|---|
| *event* | Event: RUNTIME_ERROR_EVENT, DELETE_ARTIFACT_EVENT, DEPLOY_ARTIFACT_EVENT, or EXPORT_IMPORT_EVENT. |
| *contact_type* | Contact type: EMAIL, PHONE, FAX, or SMS. |
| *value* | Email address or phone number. |

### Example

The following example removes EMAIL contact jane.doe@example.com from RUNTIME_ERROR_EVENT notifications.

```
removeContactFromNotification('RUNTIME_ERROR_EVENT', 'EMAIL', 'jane.doe@example.com')
```

## updateEvent

Command Category: MFT Contact Commands

Use with WLST: Online or Offline

### Description

Enables or disables a specific event notification. Event notifications are disabled by default. In the event of an unsupported operation, the command returns a WLSTException.

### Syntax

```
updateEvent(event, enabled)

updEvt(event, enabled)
```

| Argument | Definition |
|---|---|
| *event* | Event: RUNTIME_ERROR_EVENT, DELETE_ARTIFACT_EVENT, DEPLOY_ARTIFACT_EVENT, or EXPORT_IMPORT_EVENT. |
| *enabled* | Enabled: TRUE or FALSE. |

### Example

The following example disables DELETE_ARTIFACT_EVENT notifications.

```
updateEvent('DELETE_ARTIFACT_EVENT', 'FALSE')
```

# MFT Archive and Restore Commands

Use the MFT Archive and Restore commands, listed in Table 5-10, to archive and restore runtime instances or file system data based on criteria such as status and date ranges.

***Table 5-10 MFT Archive and Restore WLST Commands***

| Use this command... | To... | Use with WLST |
|---|---|---|
| archiveInstanceData or `arcData` | Archive runtime instances based on specified criteria. | Online or Offline |
| restoreInstanceData or `resData` | Restore previously archived runtime instances. | Online or Offline |
| archivePayloads or `arcPLs` | Archive file system data corresponding to runtime instance data based on specified criteria. | Online or Offline |
| restorePayloadsByName or `resPLbyN` | Restore previously archived file system data by the zip file name. | Online or Offline |
| restorePayloadsByPrefix or `resPLbyP` | Restore previously archived file system data by the zip file name prefix. | Online or Offline |

## archiveInstanceData

Command Category: MFT Archive and Restore Commands

Use with WLST: Online or Offline

### Description

Archives runtime instances based on specified criteria. Optionally archives file system data. In the event of an unsupported operation, the command returns a `WLSTException`.

Before you can run this command, you must create an MFT archive directory on the database server using the following SQL commands:

```
CREATE DIRECTORY MFT_DIR AS path;
GRANT READ, WRITE ON DIRECTORY MFT_DIR TO MFT_STB;
```

For more information about the database and the schema owner (MFT_STB), see *Installing and Configuring Managed File Transfer*.

### Syntax

```
archiveInstanceData(arguments)

arcData(arguments)
```

| Argument | Definition |
|---|---|
| *archiveFileName* | Name of the runtime instance archive file to be saved to the defined MFT archive directory. The default extension is `.dmp`. This argument is required. |

| Argument | Definition |
|---|---|
| *startDate* | Timestamp in format `dd-MM-yyyy H:m:s:S`. Default is no date and time specified. Expressions are not supported. |
| *endDate* | Timestamp in format `dd-MM-yyyy H:m:s:S`. Default is the current date and time. Expressions are not supported. |
| *batchId* | Identifier in the output of a previous `archiveInstanceData` command. Default input is an empty string. Use to rerun a tested or failed `archiveInstanceData` from the point of failure or to run archivePayloads. |
| *status* | Status: `*` for all statuses, `C` for `COMPLETED`, `F` for `FAILED`, or `A` for `ACTIVE` (in progress). Default is `C`. You can specify multiple comma-separated statuses. |
| *testMode* | Whether what to archive is tested but not archived: `TRUE` or `FALSE`. Default is `TRUE`. Therefore, to actually archive runtime instances, you must explicitly set `testMode=FALSE`. |
| *comments* | Comment string. Default is an empty string. |
| *runInSync* | Whether to run immediately, synchronously, and return (`TRUE`), or run in the background, asynchronously, allowing execution of other WLST commands (`FALSE`). Default is `FALSE`. |
| *fsArchiveFolderPath* | Path to the folder where the payload archive `.zip` file is saved. Default is an empty string, which means no payloads are archived. This is independent of the runtime instance archive file and the MFT archive directory. |

### Example

The following example runs in test mode (does not archive anything) and in the background (does not block WLST commands while running) for all instances with status `COMPLETED`, and provides statistics such as how many instances or payloads would be archived.

```
archiveInstanceData()
```

Here is an example of the output:

```
Total no. of instances to be archived: 105.
Total no. of payloads to be archived: 105.
Nothing has been archived as test mode was TRUE.
To archive these records, run the same command
by passing batch id 546781 and test mode as FALSE.
```

The following example runs in test mode and in the background for instances before March 31st, 2013 with status `COMPLETED`, and provides archive statistics. Note that when the *`startDate`* argument is omitted, the *`endDate`* argument must be explicitly labeled.

```
archiveInstanceData(endDate='31-03-2013 00:00:00:00')
```

The following example archives instances from February 1st, 2013 to March 31st, 2013 with status `COMPLETED` to the file `mft.dmp`, and provides archive statistics. Note that when the *`startDate`* and *`endDate`* arguments are both included, labels may be omitted.

```
archiveInstanceData(archiveFileName='mft.dmp','01-02-2013 00:00:00:00', '31-03-2013
00:00:00:00', testMode='FALSE')
```

The following example archives instances from February 1st, 2013 to March 31st, 2013 with status `ACTIVE` or `FAILED` to the file `mft.dmp`, and provides archive statistics. Because `runInSync='TRUE'` is specified, the archive occurs immediately and blocks further WLST commands until completion. Because `runFSArchive='TRUE'` is specified, file system data is also archived.

```
archiveInstanceData(archiveFileName='mft.dmp','01-02-2013 00:00:00:00', '31-03-2013
00:00:00:00', testMode='FALSE', status='A,F',
runInSync='TRUE', fsArchiveFolderPath='2013-01-02-MftArchive')
```

The following example runs a previously tested or failed archive with batch ID `546781` from the point of failure.

```
archiveInstanceData(archiveFileName='mft2.dmp',batchId='546781', testMode='FALSE')
```

## restoreInstanceData

Command Category: MFT Archive and Restore Commands

Use with WLST: Online or Offline

### Description

Restores previously archived runtime instances. In the event of an unsupported operation, the command returns a `WLSTException`.

### Syntax

```
restoreInstanceData(arguments)
```

```
resData(arguments)
```

| Argument | Definition |
|---|---|
| *`archiveFilePath`* | Name of the archive file in the MFT archive directory to be restored. For details about how this directory is defined, see archiveInstanceData. The default extension is `.dmp`. This argument is required. |
| *`fileNamePrefix`* | String to identify the file name prefix of the corresponding payload archive files, usually the batch ID. You must have run archivePayloads with the archiveInstanceData batch ID first. No default. |
| *`fsFolderPath`* | String to identify the path to the directory where the corresponding payload archive files are stored. No default. |
| *`runInSync`* | Whether to run immediately, synchronously, and return (`TRUE`), or run in the background, asynchronously, allowing execution of other WLST commands (`FALSE`). Default is `FALSE`. |

### Example

The following example restores runtime instance data previously archived by an archiveInstanceData command. The corresponding payload archive is restored from `/tmp/mft/546781.zip`. This command runs in the background.

```
restoreInstanceData(archiveFilePath="mft_runtime_archive.dmp",
fileNamePrefix="546781", fsFolderPath="/tmp/mft")
```

## archivePayloads

Command Category: MFT Archive and Restore Commands

Use with WLST: Online or Offline

### Description

Archives file system data corresponding to runtime instance data based on specified criteria. In the event of an unsupported operation, the command returns a `WLSTException`.

### Syntax

```
archivePayloads(arguments)

arcPLs(arguments)
```

| Argument | Definition |
|---|---|
| *batchId* | String to identify the archive criteria. No default: obtain a `batchId` by running archiveInstanceData first. This argument is required. |
| *archivePath* | Path to the directory to which the *batchId*`.zip` archive is saved. This directory must exist. A very large archive may be saved in multiple files named *batchId_n*`.zip`. This argument is required. |
| *runInSync* | Whether to run immediately, synchronously, and return (TRUE), or run in the background, asynchronously, allowing execution of other WLST commands (FALSE). Default is FALSE. |

### MBean Properties

For details about how to set MBean properties using Enterprise Manager, see Setting System MBean Properties for MFT WLST Commands.

| Property | Definition |
|---|---|
| *FS_ARCHIVE_MAX_SIZE* | String to set a size limit in MB for archive zip files. The default value is zero, which sets no limit. |
| *FS_ARCHIVE_MAX_FILES_PER_ZIP* | String to set a limit on the number of files an archive zip file can contain. The default value is zero, which sets no limit. |

### Example

The following example archives file system data corresponding to the runtime instance data previously archived by an archiveInstanceData command with the batch ID `546781`. The archive is saved to `/tmp/mft/546781.zip` and runs immediately.

```
archivePayloads(batchId='546781',archivePath='/tmp/mft',runInSync='TRUE')
```

## restorePayloadsByName

Command Category: MFT Archive and Restore Commands

Use with WLST: Online or Offline

### Description

Restores previously archived file system data. In the event of an unsupported operation, the command returns a `WLSTException`.

### Syntax

```
restorePayloadsByName(arguments)

resPLbyN(arguments)
```

| Argument | Definition |
|---|---|
| *fileNames* | String to identify the names of the `.zip` files to restore. No default. This argument is required.<br>If the `.zip` file names have been changed to something other than the batch IDs, you must specify the new file names instead of the batch IDs for this argument.<br>For an archive that consists of multiple *batchId_n*`.zip` or *prefix_n*`.zip` files, you can use the restorePayloadsByPrefix command instead. |
| *folderPath* | Path to the directory in which the `.zip` archives were saved. This argument is required. |
| *runInSync* | Whether to run immediately, synchronously, and return (`TRUE`), or run in the background, asynchronously, allowing execution of other WLST commands (`FALSE`). Default is `FALSE`. |

### MBean Properties

For details about how to set MBean properties using Enterprise Manager, see Setting System MBean Properties for MFT WLST Commands.

| Property | Definition |
|---|---|
| *FS_RESTORE_OVERWRITE_EXISTING* | String to determine whether a restore overwrites files having the same name. The default value is `TRUE`. |

### Example

The following example restores file system data corresponding to the runtime instance data previously archived by an archivePayloads command with the batch ID `546781`. The archive is restored from `/tmp/mft/546781.zip` and runs immediately.

```
restorePayloadsByName(fileNames='546781',folderPath='/tmp/mft',runInSync='TRUE')
```

## restorePayloadsByPrefix

Command Category: MFT Archive and Restore Commands

Use with WLST: Online or Offline

### Description

Restores previously archived file system data. In the event of an unsupported operation, the command returns a `WLSTException`.

### Syntax

```
restorePayloadsByPrefix(arguments)
```

```
resPLbyP(arguments)
```

| Argument | Definition |
|---|---|
| *fileNamePrefix* | String to identify the `.zip` files to restore, usually the batch ID. No default: obtain a `batchId` by running archivePayloads first. This argument is required.<br>If the `.zip` file names have been changed to something other than the batch ID, you must specify the new file name instead of the batch ID for this argument.<br>If you rename one *`prefix_n`*`.zip` file you must rename all, without changing the *`_n`* portion, or this command will not restore the entire archive. |
| *folderPath* | Path to the directory in which the `.zip` archives were saved. This argument is required. |
| *runInSync* | Whether to run immediately, synchronously, and return (`TRUE`), or run in the background, asynchronously, allowing execution of other WLST commands (`FALSE`). Default is `FALSE`. |

### MBean Properties

For details about how to set MBean properties using Enterprise Manager, see Setting System MBean Properties for MFT WLST Commands.

| Property | Definition |
|---|---|
| *FS_RESTORE_OVERWRITE_EXISTING* | String to determine whether a restore overwrites files having the same name. The default value is `TRUE`. |

### Example

The following example restores file system data corresponding to the runtime instance data previously archived by an archivePayloads command with the batch ID `546781`. The archive is restored from `/tmp/mft/546781.zip` and runs immediately.

```
restorePayloadsByPrefix(fileNamePrefix='546781',folderPath='/tmp/
mft',runInSync='TRUE')
```

# MFT Purge Commands

Use the MFT Purge commands, listed in Table 5-11, to purge runtime instances or file system data based on criteria such as status and date ranges.

***Table 5-11 MFT Purge WLST Commands***

| Use this command... | To... | Use with WLST |
|---|---|---|
| purgeInstanceData or `prgData` | Purge runtime instances based on specified criteria. Optionally purges file system data. | Online or Offline |
| purgePayloads or `prgPLs` | Purge file system data corresponding to runtime instance data based on specified criteria. | Online or Offline |

## purgeInstanceData

Command Category: MFT Purge Commands

Use with WLST: Online or Offline

### Description

Purges runtime instances based on specified criteria. Optionally purges file system data. In the event of an unsupported operation, the command returns a `WLSTException`.

### Syntax

```
purgeInstanceData(optional_arguments)

prgData(optional_arguments)
```

| Argument | Definition |
|---|---|
| *startDate* | Timestamp in format `dd-MM-yyyy H:m:s:S`. Default is no date and time specified. Expressions are not supported. |
| *endDate* | Timestamp in format `dd-MM-yyyy H:m:s:S`. Default is the current date and time. Expressions are not supported. |
| *batchId* | Identifier in the output of a previous `purgeInstanceData` command. Default input is an empty string. Use to rerun a tested or failed `purgeInstanceData` from the point of failure or to run purgePayloads. |

| Argument | Definition |
|---|---|
| *status* | Status: * for all statuses, C for COMPLETED, F for FAILED, or A for ACTIVE (in progress). Default is C. You can specify multiple comma-separated statuses. |
| *testMode* | Whether what to purge is tested but not purged: TRUE or FALSE. Default is TRUE. Therefore, to actually purge runtime instances, you must explicitly set testMode=FALSE. |
| *comments* | Comment string. Default is an empty string. |
| *runInSync* | Whether to run immediately, synchronously, and return (TRUE), or run in the background, asynchronously, allowing execution of other WLST commands (FALSE). Default is FALSE. |
| *runPayloadPurge* | Whether to also purge file system data corresponding to runtime instance data: TRUE or FALSE. Default is FALSE. To purge file system data separately, use purgePayloads. |

### Example

The following example runs in test mode (does not purge anything) and in the background (does not block WLST commands while running) for all instances with status COMPLETED, and provides statistics such as how many instances or payloads would be purged.

```
purgeInstanceData()
```

Here is an example of the output:

```
Total no. of purgeable instances: 105.
Total no. of purgeable payloads: 105.
Nothing has been purged as test_mode was TRUE.
To purge these records, run the same command
by passing batchId 546781 and test mode as FALSE.
```

The following example runs in test mode and in the background for instances before March 31st, 2013 with status COMPLETED, and provides purge statistics. Note that when the *startDate* argument is omitted, the *endDate* argument must be explicitly labeled.

```
purgeInstanceData(endDate='31-03-2013 00:00:00:00')
```

The following example purges instances from February 1st, 2013 to March 31st, 2013 with status COMPLETED, and provides purge statistics. Note that when the *startDate* and *endDate* arguments are both included, labels may be omitted.

```
purgeInstanceData('01-02-2013 00:00:00:00', '31-03-2013 00:00:00:00',
testMode='FALSE')
```

The following example purges instances from February 1st, 2013 to March 31st, 2013 with status ACTIVE or FAILED, and provides purge statistics. Because runInSync='TRUE' is specified, the purge occurs immediately and blocks further WLST commands until completion. Because runPayloadPurge='TRUE' is specified, file system data is also purged.

```
purgeInstanceData('01-02-2013 00:00:00:00', '31-03-2013 00:00:00:00',
testMode='FALSE', status='A,F', runInSync='TRUE',
runPayloadPurge='TRUE')
```

The following example runs a previously tested or failed purge with batch ID `546781` from the point of failure.

```
purgeInstanceData(batchId='546781', testMode='FALSE')
```

## purgePayloads

Command Category: MFT Purge Commands

Use with WLST: Online or Offline

### Description

Purges file system data corresponding to runtime instance data based on specified criteria. In the event of an unsupported operation, the command returns a `WLSTException`.

### Syntax

`purgePayloads(arguments)`

`prgPLs(arguments)`

| Argument | Definition |
|---|---|
| *batchId* | String to identify the purge criteria. No default: obtain a `batchId` by running purgeInstanceData first. |
| *detailedAudit* | Whether each purged file is audited: `TRUE` or `FALSE`. Default is `TRUE`. |
| *runInSync* | Whether to run immediately, synchronously, and return (`TRUE`), or run in the background, asynchronously, allowing execution of other WLST commands (`FALSE`). Default is `FALSE`. |

### MBean Properties

For details about how to set MBean properties using Enterprise Manager, see Setting System MBean Properties for MFT WLST Commands.

| Property | Definition |
|---|---|
| *FS_PURGE_BATCH_SIZE* | String to determine the batch size for reading payload paths. The `purgePayloads` command reads the payload paths from the database in batches to avoid running out of memory. The default value is `5000`. |
| *FS_PURGE_TEMP_TABLES_LIFE* | String to determine the minimum age in days of temporary tables that are deleted after a payload purge. The default value is `30`. |
| *FS_PURGE_THREAD_COUNT* | String to determine how many threads per node are used for performing the payload purge. The default value is `1`. |

### Example

The following example purges file system data corresponding to the runtime instance data previously purged by a purgeInstanceData command with the batch ID `546781`. Detailed audits of purged files are performed, and the purge runs in the background.

```
purgePayloads(batchId='546781')
```

The following example purges file system data corresponding to the runtime instance data previously purged by a purgeInstanceData command with the batch ID `546781`. Detailed audits of purged files are **not** performed, and the purge runs in the background.

```
purgePayloads(batchId='546781', detailedAudit='FALSE')
```

The following example purges file system data corresponding to the runtime instance data previously purged by a purgeInstanceData command with the batch ID `546781`. Detailed audits of purged files are performed, and the purge occurs immediately.

```
purgePayloads(batchId='546781', detailedAudit='TRUE', runInSync='TRUE')
```

## Setting System MBean Properties for MFT WLST Commands

Some MFT WLST commands have associated properties you can set to refine the operations they perform.

The steps for this process are:

1. Log in to the Oracle Enterprise Manager console.
2. In the Target Navigation pane, expand the Weblogic Domain node.
3. Select the domain on which the Oracle WebLogic Server managed server dedicated to Oracle Managed File Transfer is installed.

   For example, the domain might be `soainfra` or `base_domain`.
4. Right-click on the domain and select **System MBean Browser**.
5. Click the Find icon.
6. Type `MFTConfig` in the **MBean Name** text box. Press Enter.

   The Application Defined MBeans: MFTConfig: mft page appears.
7. Click the **Operations** tab.
8. Click **addProperty**.

   The Operation: addProperty page appears.
9. Type the property name in the **key** text box. Type the property value in the **value** text box. Typing a **comment** is optional. Click **Invoke**.
10. To verify that the property has been added:
    a. Click **Return**.
    b. Click the **Attributes** tab.

**c.** Click **Properties**.

The Attribute: Properties page appears. The property you added should be listed.